Effective simulation studies using domain specific simulation building blocks

Edwin Valentin

Effective simulation studies using domain specific simulation building blocks

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 19 april 2011 om 15.00 uur

door

Edwin Cristiaan VALENTIN
Baccalaureus logistiek en economie

geboren te Delft

Dit proefschrift is goedgekeurd door de promotoren:

Prof. dr. H.G. Sol
Prof. dr. ir. A. Verbraeck

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof. dr. H.G. Sol, Technische Universiteit Delft, promotor
Prof. dr. ir. A. Verbraeck, Technische Universiteit Delft, promotor
Prof. dr. N.M. van Dijk, Universiteit van Amsterdam
Prof. dr. ir. L.A. Tavasszy, Technische Universiteit Delft
Prof. dr. ir. W.A.H. Thissen, Technische Universiteit Delft
Prof. dr.ir. J.G.A.J van der Vorst, Universiteit Wageningen
Dr. I. Wenzler, Technische Universiteit Delft
Prof. dr. F.M.T. Brazier, Technische Universiteit Delft, reservelid

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, Den Haag
Valentin, Edwin Cristiaan
Effective simulation studies using domain specific simulation building blocks
Proefschrift Delft. – Met Index, lit. opg., Nederlandse samenvatting
Trefw.: discrete event simulation, building blocks, modeling

Cover illustration: Edwin Valentin
Printing: Gildeprint Drukkerijen
English editor: Miranda Aldham-Breary

Preface and Acknowledgements

In the past 13 years that I have been working in the domain of simulation modeling I have discussed the challenges in the field with a lot of people with a wide variety of background. To mention two extremes: With my friends I mainly spoke about the use of providing insight in queues, resulting in practical jokes were my advice was requested to help selecting the best queue at our local supermarket. Just as practical, but more interesting were the discussions with real life problem owners. “Are two berths sufficient?”, “Is this silo large enough?” or “Can we improve the production by 10%?” are just a couple of the many real life problems I have tackled.

This dissertation could not have been developed without these real life problems, but if you are reading this dissertation to find the answers to the questions then I have to disappoint you. This dissertation focuses around the question whether a different way of performing simulation studies is more effective. Years I have been claiming “Yes, building blocks will rule the world”, but putting that down in the right words have given me more headaches than all my simulation studies, modeling problems and unexplainable errors together. And thanks to all the help, support, motivation and inspiration I have received an end has been put to this pending challenge of writing a dissertation.

“Schrijven is schrappen”[1] has never been my best point, but I owe it to both my promotores Henk Sol and Alexander Verbraeck that they kept on telling me to stick to what I can and know. If I look back to some of the documents I have asked them to review, then it is a miracle they did not kick me out. Instead they kept on helping and guiding me towards a complete research and not just two dozen of fun simulation studies without any relation.

On the other hand, I have had the opportunity to participate in these simulation studies and apply there my ideas and suggestions. I want to thank my contacts at the different institutes and (simulation) companies as well as the students and colleagues that actively worked with me in these projects, especially Adrien Leleu, Alessandro Nati, Annemarie Corver, Corné Versteegt, Dave Sturrock, Dennis Pegden, Freeke Heijman, Gregory Piot, Igor Mayer, Jeroen Korf, Klaas Pieter van Til, Martijn Riesenkamp, Patrick Blom, Piero Silva, Rachid Maghnouji, Remmert van der Wal, Rienk Bijlsma, Rogier van der Hee, Sicco Steijaert, Tim Holt, Vincent de Gast, Wieke Bockstael and Yvo Saanen.

I have had the opportunity to share the results of these simulation studies and intermediate results of the use and design for simulation building blocks in different scientific communities. Each group learned and showed me something different ranging from the friends from Systeemkunde, the similar minded researchers of the BETADE research program and PhD consortia to

[1] Kill your darlings

the experts at simulation conferences such as WSC and ESS. To most of the people I have met I owe thanks, especially the persons that over the years kept interested, supporting and motivating in addition to previously mentioned names: Ajantha Dehanayake, Bob Briggs, Dick Nance, Els van de Kar, Gert Jan de Vreede, Gwen Kolfschoten, Hans Vangheluwe, Jaco Appelman, Jerry Banks, Jessica Chen, Glenn Drake, Jon Philips, Jurgen van Grinsven, Marielle den Hengst, Miranda Aldham-Breary, Nadia Ayad, Osman Balci, Peter Jacobs, Peter Keen, Roy Chin, Sabrina Rodriques, Stijn Pieter van Houten, Tamrat Tewoldeberhan, Wander van der Berg, Wenlong Zhao and Yan Wang.

It took me several years longer than the 4 years that are provided normally by the university. Therefore I want to thank my employers Delft University, Systems Navigator and Accenture for providing me with time, facilities and trust to finish off my research and dissertation in the time I needed.

Finally all this wouldnot have succeeded if my friends and family did not provide the support and distractions that were necessary to empty and fill my head. I cannot think of a better word for standing by me over the years than: “Thanks”.

Edwin Valentin

2011.

Table of contents

1. Effective problem solving using discrete event simulation

1.1 The use of models in problem solving

Problem owners need a good insight into their systems to be able to make decisions for improving or changing the systems. They gather this insight using models of the current systems and of possible alternatives. Any models that problem owners use in their process of problem solving are reduced representations of reality. Ackoff (1962) states that the process of problem solving consists of two phases. In the first phase the problem or issue encountered in a system is analyzed based on a model of the current system. This model is used to identify a variety of potential solutions to improve the system and thus reduce or remove the problem, or address the issue. In the second phase the solutions are represented by models. These models are evaluated and judged on a set of performance indicators identified by the problem owners, which are often compared to the values of the same performance indicators obtained by studying the model of the current system. The outcome of the second phase is one selected solution that is applied to the existing system to result in a new system that no longer presents the observed problem.

The complete process of problem solving as identified by Ackoff consists of six sequential steps. These six steps are (after Ackoff, 1962):

Phase one:

1. *formulating the problem*
2. *constructing the model*
3. *analysis with the model*

Phase two:

4. *deriving solutions from the model*
5. *analyzing solutions and selecting a solution to be implemented*
6. *implementing the solution*

The process of problem solving is aimed at finding a solution that best fits the requirements of the problem owners. Usually, problem owners can easily define a set of possible solutions for a problem, but comparing the different solutions is difficult due to the lack of insight into the potential results. Therefore, Ackoff introduces and uses models to represent the original system and potential solutions. The models of the different variants of the system will be compared by the problem owners, so they can make a decision which solution to implement in reality. The problem owners will judge the quality of solutions on performance indicators that can be obtained from their models. It is important within the process of problem solving that the models of the

problem situation and the solutions provide the same set of performance indicators. In practice, one model of the current situation is constructed, which is adjusted to analyze the alternatives. In this way the models provide the same performance indicators, and solutions can be compared objectively. The level of success of using models in problem solving can be defined by the ability of problem owners to base their decisions on the performance indicators obtained by studying the models. These performance indicators can be quantitative, but also qualitative judgments by the problem owner or by experts.

Identifying the best solution for a certain system requires that all solutions are identified and modeled. Developing models of all solutions for real-world problems is impossible. Simon (1969) introduces the concept of *bounded rationality* for the complex multi-actor process of problem solving, acknowledging the fact that it is impossible to see the complete solution space.

Models do not reflect the complete system with all its aspects, but rather they are an abstraction that is applied to include only those aspects of the system that are important for the solving of the problem. Mitroff and Sagasti (1973) extend Ackoff's process of problem solving by defining the use of several types of models to support problem solving. The first type of models is the *conceptual model* in which the structure, concepts and boundaries of the system are defined. The second type of models is the *empirical model*. Empirical models are used to represent the system within the structure, concepts and boundaries of the conceptual model, to provide insight for the problem owners into that particular system configuration. Sol (1982) adds that a conceptual model defines the *language* that is used to *instantiate* the empirical models and the conceptual model defines the *view* that model developers apply to represent the system using empirical models. The distinction between a conceptual and an empirical model is important, because choices for the conceptual model might limit the scope of the empirical model. During the process of problem solving the conceptual model is used to describe the structure and concepts for all empirical models that represent current and possible future systems.

An example of the use of different empirical models in problem solving is the following situation: a computer hardware factory produces boxed computers at a fixed interval. These boxes need to be sealed by a sealing machine. The number of boxes to be produced will increase in the coming year and therefore the operation manager of the factory foresees a problem in the sealing department. Boxes arrive via a conveyor belt, are sealed by a machine, continue on a conveyor belt, are placed on pallets after which a forklift truck delivers the pallets to the warehouse. If a pallet is full, a box has to wait for a new pallet. If the second conveyor is full, the sealing machine cannot continue. One operator moves between the two sealing machines, replaces pallets once a forklift moves a full pallet away, and repairs the sealing machines if they break down.

A conceptual model that abstracts the above description into the objects in the system and the flow of the boxes through the system is given in Figure 1.1.

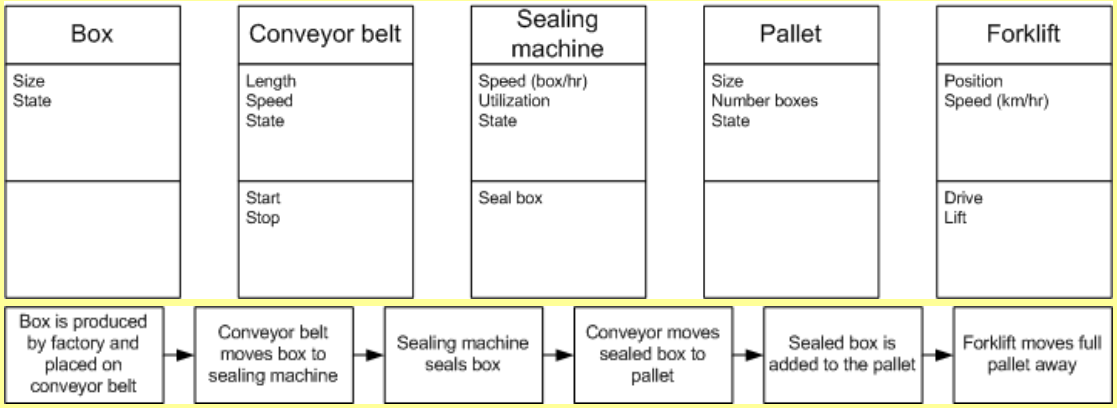


Figure 1.1: Conceptual model of problem system

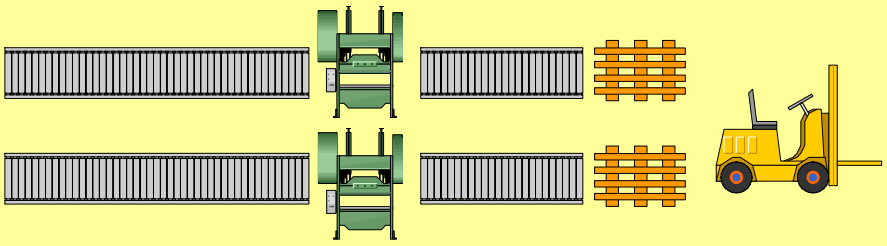


Figure 1.2: Empirical model of problem system; layout drawing

Figure 1.2 shows a drawing of the system. In this empirical model several elements have not been modeled, for example the sensors on the conveyor belts and the equipment to put the sealed boxes on top of the pallet. Figure 1.3 is a model using a spreadsheet to calculate the expected throughput and utilization of the sealing machines, assuming that the factory is operating 5 days a week for 16 hours a day.

	boxes per hour	boxes per week
Input	92	**7360**
Rate seal machine A	75	6000
Rate seal machine B	35	2800
		8800
Expected utilization		
Seal machine A		84%
Seal machine B		84%

Figure 1.3: Empirical model of problem system; spreadsheet calculation

These two empirical models show the same system, but each of the models gives the problem owners a different type of insight into the operation of the system and what they can expect if they invest extensions or alternative solutions.

Figure 1.4 is a representation of the process of problem solving, derived from the work of Ackoff (1962), Simon (1969) and Sol (1982). This figure clearly shows the two phases introduced by Ackoff (1962), i.e. first analyze before developing models of alternative solutions. The iteration loops of Simon (1969) are also included in the problem solving cycle and consist of validating the empirical model and models of alternative solutions using a consistency check between alternative solutions and the conceptual model, and *ex ante* and *ex post* evaluations of solutions. Finally, Figure 1.4 shows that the analysis of the problem system is based on empirical models developed from a conceptual model as described by Sol (1982). A distinction is made in the figure between process steps that result in new models and steps that are performed for checking and evaluating the models or systems. The first type of steps are represented with uni-directional arrows, the latter type of steps with bi-directional arrows.

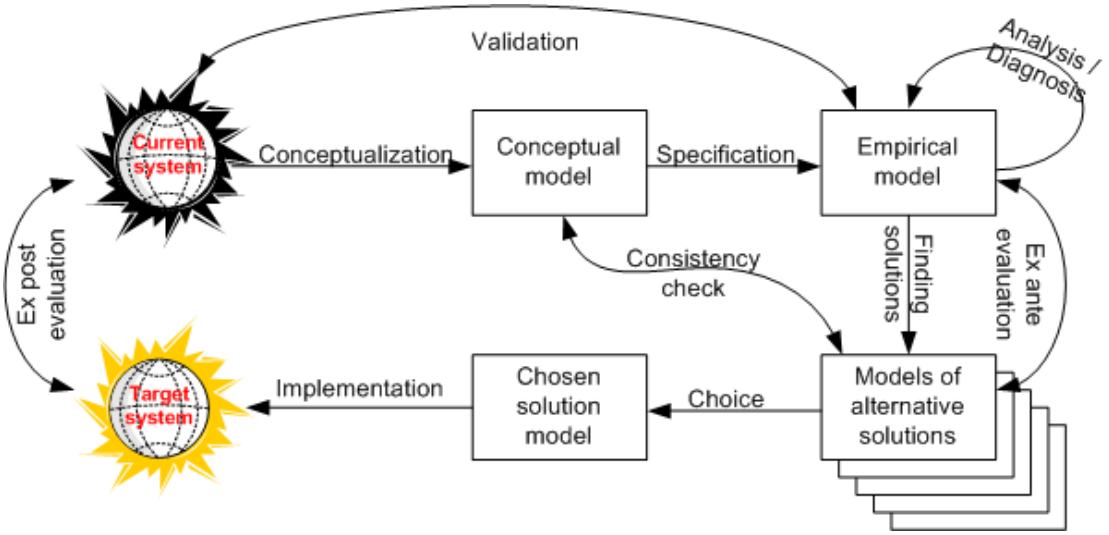


Figure 1.4: Process of problem solving derived from Ackoff (1962), Simon (1969) and Sol (1982)

1.2 Research scope: Problem solving support by discrete event simulation

The process as represented in Figure 1.4 is generic and can be supported by different types of conceptual and empirical models. Conceptual models provide the generic structure and boundaries of the system as it will be studied, and can for instance be represented by a set of processes and objects that are considered to be important to address the problem or issue. These can be described using e.g., text, object models and flow charts. Empirical models represent the system or solution and can be represented using many different types of models. Examples of quantitative empirical models are mathematical equations or spreadsheets, for example to be able to calculate cost and turnover of a system. These models are *static* representations of the system, showing one moment or state of the system.

Within this research we focus on problem solving supported by *dynamic* empirical representations of the system, and more particularly “discrete-event” simulation models.

Simulation models are models that are not just a fixed representation of the system at one moment in time, but rather they show the system in “operation” Simulation models are commonly applied in systems where static calculations using spreadsheets or queuing theory are insufficient. Standard queuing theory or spreadsheet calculations cannot be applied, and solutions tend to be non-linear. Simulation models allow us to show the effects of interactions over a time period and thus enable problem owners to gather insight into the dynamic aspects of their system.

Simulation models can be developed using different formalisms (Zeigler et al, 2000; Vangheluwe and Vansteenkiste, 1997). The most common types of formalisms are continuous simulation and discrete event simulation. In continuous simulation the state changes of the system are calculated by solving a set of differential equations over time. In discrete event simulation the state changes of the system take place at fixed moments in time.

A *simulation model* of the sealing department of the computer factory will provide additional insights into the system that did not appear in the models of Figure 1.2 or Figure 1.3. If the forklift is delivering the pallet, the conveyor belt and the sealing machine might have to halt. At the same moment an operator could be working on a breakdown at the second sealing machine. Figure 1.5 is a drawing of a possible state of the system at a random moment during the week. The static empirical models of Figure 1.2 or Figure 1.3 do not give any insight into these possible states of the system. Depending on the distance the forklift has to travel or the breakdown interval of the machines this particular state could occur frequently and thus the performance of the factory will not be as high as concluded based on the spreadsheet model of this system.

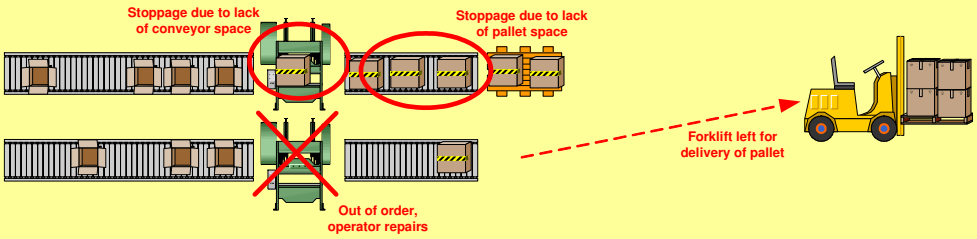


Figure 1.5: Empirical model of problem system; possible state during operation

Applying simulation models in problem solving will improve the insight of the problem owners into the system, but it might also lead to an evaluation of more alternative solutions. For example, after having seen a run of the simulation model, the travel distance of the forklift becomes important as well as the breakdown interval of the machines. New solutions can include new layouts, more buffer spaces between the machines and the pallets, and the allocation of priorities to the operator.

Discrete event simulation enables model developers to represent the *behavior* of a system and its elements over time, to include randomness and variance, and to calculate performance indicators that take into account the effect of time and variance. The suitability of discrete event simulation in problem solving can be summarized as follows (Shannon, 1975; Law and Kelton, 1999; Banks, 1999; Kelton et al, 2003). Discrete event simulation provides us with:

- the ability to explicitly model the dynamic behavior of a system over time, and thereby gaining insights into the way the system functions;
- the ability to obtain quantitative results from running the simulation model, both for the current situation and for potential future situations;
- the ability to visualize a system during its operation to observe bottlenecks or shortcomings;
- the ability to visualize the effects of different courses of actions for a system by observing adjusted operations;
- the ability to imitate a system even though the data is incomplete; imitation is achieved by a reduction in complexity or by using assumptions with regards to data;
- the ability to communicate the working of the system to different actors involved;
- the ability to include stochastic effects in the models and in the calculated performance indicators.

1.3 Elements of a discrete event simulation model

The models presented in the example of the computer factory are empirical models that are based on an abstraction of reality. This abstraction excludes certain types of equipment, such as the sensors at the conveyor belt. In dynamic models, like simulation models, the processes in the system are an important part of the model. Also, not all processes that can be identified in reality will be included in the models; some of these processes will be left out (abstraction) or simplified, which is called *reduction* in simulation modeling. One of the processes for which reduction was applied in the simulation model is the coffee break of the operator, another is the process of feeding the seal machine with new tape.

The abstractions that apply to empirical models, both the abstraction in scope or equipment and the abstraction in processes, are described in the conceptual models. A conceptual model describes how the system will be represented and what parts of the system will be left out of the simulation study. In the conceptual model the system is often divided into smaller parts that have relations and together represent the complete system. We follow Flood and Carson (1988) who define a system as "*an assembly of elements related in an organized whole*" (p7). According to Flood and Carson an "*element may be anything that is discernible by a noun or a noun phrase that*

all informed observers would agree exists" (p7). The "organized whole" contains a set of relations that these entities have with each other (Sol, 1982). The relations can be static dependencies or dynamic interactions which affect the behavior of the system. The different elements in a system will be identified by decomposition of the system.

In so-called discrete-event simulation that provides a representation a system with a focus on the logical and physical flows, the processes and the elements are instantiated in a simulation model to represent the system dynamically. The descriptions of the elements and the processes in a simulation model are static, until the processes are triggered and executed. The simulation model will take its dynamic behavior from the system in a simulation environment. A **simulation environment** is a set of one or more applications that support the model developer in instantiating a simulation model to represent a system and to execute the processes for a defined time frame (Nance, 1993). A simulation environment will control the clock in the simulation model and trigger events as discrete scheduled events for the duration of the simulation experiment.

Figure 1.6 shows a static representation of the processes of the simulation of the sealing department at a computer factory. This simulation model was developed in the simulation environment Arena which provides a flow chart approach to describe the processes in the system. During a simulation run, the model will show the state of the system at exact moments in time, for example the number underneath the first process "Transport via conveyor to machine" will show the number of boxes present in that process.

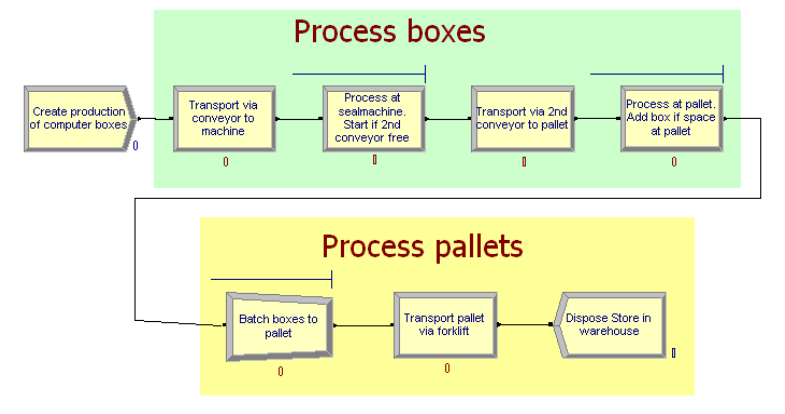


Figure 1.6: Representation of empirical model for processes of problem system in simulation environment Arena

Figure 1.7 shows a static representation of the elements in the system. In the top of the figure there is a picture of the equipment, at the bottom a definition of the elements with their capacity and the failures that apply to the seal machines. During execution of the simulation model the top of the figure will be comparable to Figure 1.5 and thus represent the exact state of the system at a discrete event.

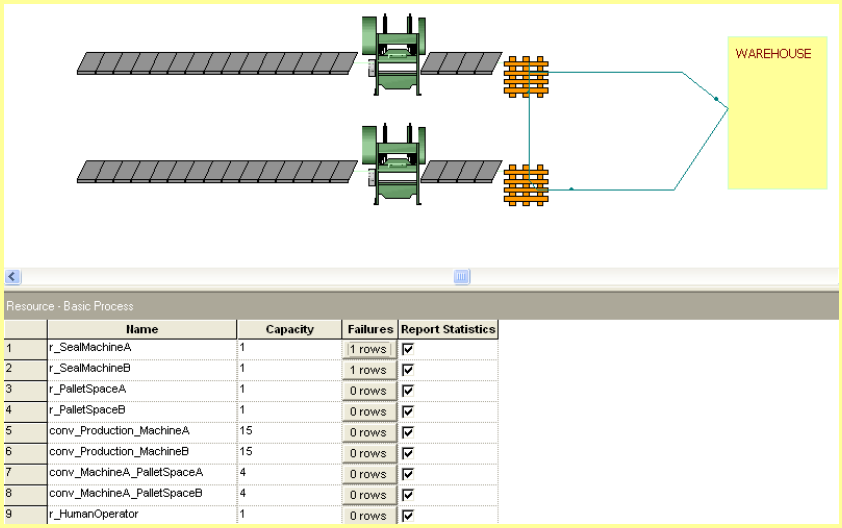


Figure 1.7: Representation of empirical model for elements of problem system in simulation environment Arena

At the end of the simulation run, statistics can be gathered comparable to the spreadsheet of Figure 1.3. The difference is that the pure spreadsheet data is a calculated prediction which does not include stoppages, breakdowns or blockages. All these system characteristics that can influence the outcome are, however, reflected in the calculated performance indicators after running the simulation model.

Figure 1.6 and Figure 1.7 are screen dumps of a simulation model developed in the simulation environment Arena (Bapat and Sturrock; 2003). Examples of other popular discrete event simulation environments are eM-Plant (Heinicke and Hickman; 2000), Witness (Mehta; 1999), Promodel (Harrell and Price; 2003), Enterprise Dynamics (Britals, 2008) and Automod (Rohrer, 2003). These simulation environments are generic and can be applied in many domains. We will refer to these as **generic simulation environments**.

The simulation environments provide certain elements to compose a simulation model. All simulation environments use different names to refer to these elements. For example, Arena and Witness use “modules”, eM-Plants uses “objects”, Enterprise Dynamics uses “atoms” and Automod and Promodel use “elements”. We will refer to these elements in a simulation environment as **model constructs**, these are the elements in a simulation environment that are used to compose a simulation model.

Model constructs are instantiated in the simulation model to represent elements of the system. An element of the conceptual model can be defined by one or more model constructs. The model constructs in the generic simulation environments provide generic representations of system elements, thus in most cases the elements that are defined in the conceptual model will be represented by a collection of model constructs that together provide a valid representation of the element in the real system (Birtwistle, 1979; Pegden et al, 1990; Balci and Nance, 1992; Banks, 2000). Each model

construct can be parameterized in a certain way, and thus two instances of a model construct can represent different system elements with (slightly) different properties or behavior.

1.4 Applying discrete event simulation models for problem solving

In this research, we follow Shannon (1975) with regard to his definition of the process of using simulation models in problem solving: "*the process of designing a model of a concrete system and conducting experiments with this model in order to understand the behaviour of a concrete system and/or to evaluate various strategies for the operation of the system*." (p.2).

Discrete event simulation studies follow the process of problem solving as it is described in Figure 1.4. Figure 1.8 is a specification for problem solving using discrete event simulation environments based on the process descriptions of Shannon (1975), Banks (1999) and Kelton et al (2003). This process is a more detailed description than the one given in Figure 1.4 and pays more attention to the development of conceptual and simulation models and the analyses that are performed with the simulation models. Following Ackoff, the process of performing a simulation study to support problem solving can also be separated into two phases. The first phase, above the dotted line in Figure 1.8, provides an analysis to the problem system using a valid model of the problem system, and the second phase, below the dotted line in Figure 1.8, evaluates solutions using simulation models of alternative systems.

The conceptualization process resulting in a conceptual model as shown in Figure 1.4 is performed for a simulation study in three process steps as shown in Figure 1.8: "Problem description", "Define conceptual model" and "Select model constructs to represent system elements". The conceptual model contains a clear boundary with the model environment to limit the need for quantitative data for the empirical model. Of course the boundaries should be so wide that the problem can be solved with the simulation study, but not much wider. The result of this process step is a structured overview of elements and processes of the system that will be included in the simulation model to be developed. This overview uses the terminology of the domain to enable the problem owner to understand the way the model developer has abstracted knowledge from the system. Model developers will make choices as to how to represent the system elements with model constructs depending on their knowledge of the simulation environment, their affinity with the problem domain and the input of the problem owner. The element definitions in the terminology of the domain are translated to the applicable model constructs of the generic simulation environment in the next process step, which is part of the so-called specification of the simulation model.

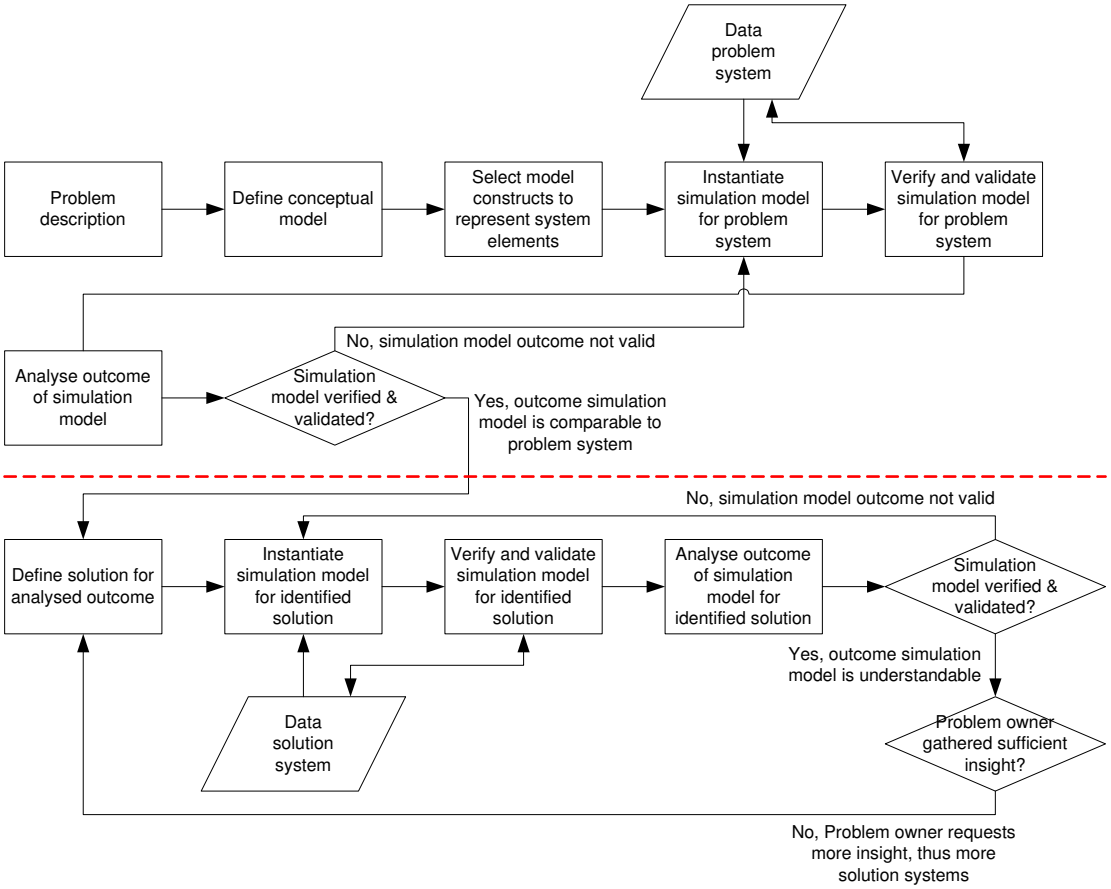


Figure 1.8: Process of a simulation study based on process descriptions of Shannon (1975), Banks (1999) and Kelton et al (2003)

The simulation model can be instantiated based on data gathered from the system and its elements using model constructs, therefore the incoming information “data problem system” at the top of Figure 1.8. The instantiation of the simulation model should be a straightforward process, because all the thinking and defining activities have been performed for the development of the conceptual models. The process of instantiating is thus the parameterization of the model constructs.

Verification of a simulation model is an activity in which the conceptual model is compared with its representation provided by a simulation model. Model developers evaluate whether the model has been derived correctly from the conceptual model during the verification process step. This process step enables modelers to find programming or parameterization errors made during model development.

During validation in a simulation study, the simulation model is compared with the real system to see whether the simulation model is a valid representation of the system. Significant differences in the outcome or behavior of the simulation model, or expectations of the problem owners that have not been met, hint at errors of the model developer in the translation of

the system to the simulation environment or to inconsistencies or flaws in data used to instantiate the simulation model.

The process “Analyze outcome of simulation model” prepares for the design of alternative solutions, and it triggers the process “Define solution for analyzed outcome” to try to find alternatives for the system to overcome or reduce the observed issues.

The identified solution should then be instantiated in the simulation model to enable further analysis and a (statistical) comparison between the original model and the solution. From Figure 1.4 it can be seen that the models of the solution systems are usually based on the same conceptual model. The process step “Instantiate simulation model for identified solution” will therefore result in a simulation model that can be compared to the original simulation model. The simulation model of the solution system can be a brand new simulation model, but most often the solution model will be based on the original simulation model, and it only includes some additional or different model constructs or it even contains the same model constructs but with different parameter values.

After the simulation model of the solution has been verified the problem owner can use the outcome to gather insight and judge the effects of the proposed solution. Once sufficient insight is gathered the simulation study will be finished. Otherwise more solutions for the problem might be defined so these can be evaluated using simulation models, until the problem owner has gathered the insight necessary for the process of problem solving.

1.5 Common challenges in simulation studies

Even though there are clearly advantages and benefits of using simulation models, many problem owners are not fully satisfied with the simulation studies (Robinson and Pidd, 1998). These problem owners cannot base their decisions on the outcome of the simulation study. **Effectiveness** of a simulation study is roughly defined as the closeness of the gathered insight and required insight of the problem owner. The gathered insight is all of the insight the problem owner gained during the simulation study regarding the problem system and possible solutions. The required insight is the insight that the problem owner, before the study, expected to gain using the simulation study to be able to successfully perform the process of problem solving and solution selection. A small gap between the provided insight and the required insight means an effective simulation study. Law and McComas (1989) state that problem owners often do not know what to expect from a simulation study, so in these simulation studies it is difficult to determine the required insight of the problem owner. Nevertheless effectiveness of a simulation study can be evaluated by interviewing the problem owners and identifying whether the gathered insight was sufficient to satisfy their needs.

Robinson and Pidd (1998) carried out interviews among providers and users of simulation models. They conclude that the main reason for not using discrete event simulation models in problem solving is uncertainty as to whether the investment into developing a simulation model and gathering valid data will result in enough “added value” for the problem owners. Often problem owners find out that the effort they have spent on the simulation study, both in time and money, does not result in the insights they need to support their process of problem solving.

Problem owners gain most insight from an analysis of the performed simulation experiments of solution systems. If a problem owner encounters insufficient insight at the end of a simulation study, then this is mostly a result of insufficient analysis of possible solutions. The problem owners claim a lack of experiments, a lack of performance indicators and a lack of trust that the simulation model represents their problem correctly. Insufficient results presented to the problem owner is caused by projects that are finished before the problem owner captures the insight. Many internal and external factors cause simulation studies to be finished before the problem owners are completely satisfied. The available budget, expectations of the problem owners and operating as a team are common causes for failure in many technical projects, including simulation studies. Robinson and Pidd (1998) observed three reasons why problem owners perceive a gap between the required insight and the gained insight, i.e. ineffective simulation studies.

One, **difficulties to handle the unlimited freedom in modeling by model developer**: the generic simulation environments offer, with their generic model constructs, a lot of freedom how a model developer represents a system. The model developer has to select the model constructs that can be used to represent a system, and configure and combine these model constructs in such a way that the system is correctly modeled. A small part of the system will be modeled by a large number of many generic model constructs. The model developer will make multiple decisions and perform a lot of actions until the model is as he envisions the best representation of the system. Mistakes can be made in every decision and every action performed. Further, a minor change to the problem system may require remodeling several system elements, using generic model constructs in an alternative way.

Two, **model developers need to be experts in multiple areas**: model developers should be multifunctional persons. In each of the activities mentioned in Figure 1.8 the model developer needs to apply a different technical skill, ranging from conceptualization to computer engineering and from database knowledge for data generation to statistics for output analysis. Further, the model developer needs to be an advanced consultant who can explain to the problem owner the scope of the simulation study and extract from the problem owner all kinds of system specific characteristics. Finally,

the model developer needs to be an expert on the domain to be able to speak the language of the problem owner. The basic skills can be learned at university, but the diversity of the model development work requires training and experience. Skills commonly found to be lacking in model developers are those dealing with statistics, conceptual modeling and implementation skills (Law and McComas, 1989; Keller et al, 1991; Sadowski and Grabau, 2000). Insufficient statistical skills cause model developers generate incorrect outcome or make ungrounded conclusions. Conceptual modeling is difficult for model developers who are not an expert in the problem system. Setting boundaries to the system requires insight and experience in a problem system. Implementation skills are required to make a valid translation to model constructs. System elements can be instantiated in simulation models in different ways and using a certain translation can limit the experiments that can be performed or require system functionality to be added.

Three, **model developers do not speak the language of the problem owner**: the problem owner and model developer defined in the conceptual models the scope of the simulation study. The model developer cannot be the person who knows everything about the system, especially at the start of the simulation study, therefore he depends heavily on the information provided by the problem owner. The model developer often has the intention to generalize the system elements to the model constructs available in the simulation environment he is working with. The conceptual models will then be composed out of generic objects as a “resource” or a “queue” while the problem owner speaks in words such as “forklift” and “high-speed stack crane”. The ‘language mismatch’ applies to all activities in the simulation study, but mainly to the initial scope of the problem system. Misinterpretation will lead in later activities to rework and extension of the scope. Fixing the issues caused by the ‘language mismatch’ will consume a lot of time, and can result in the fact that one or more of the solution systems cannot be modeled during the time allowed for the simulation study. When some experiments cannot be performed, the problem owner cannot achieve the insight requested.

The three causes of the perceived gap between required insight and achieved insight using simulation models discussed above often make it often difficult for the model developers to adjust a simulation model for a solution system (Robinson and Pidd, 1998). If simulation environments were less generic, model developers would have less modeling freedom. The generality of these simulation environments is seen as a strength by advanced model developers (Robinson and Pidd, 1998). Advanced model developers are fully aware of the generic simulation environment and like the generality, because this allows them to model a system exactly according to their preferences.

Less advanced model developers do not need full control and full power over the way they represent a system in a simulation model. These less

advanced model developers have difficulties with the large gap between the conceptual model of a problem system and the model constructs of generic simulation environments. The large gap requires the translations of elements of conceptual models into a combination of model constructs. Model developers would not have this problem if the model constructs of the simulation environment were more specific. Instead of instantiating a simulation model from model constructs like a resource or a queue the model developers should be able to instantiate a model from problem domain specific model constructs. As a result the translation from concept to model construct will be less difficult and the model developer will be able to adjust the simulation model more easily to run additional experiments to satisfy the problem owner.

1.6 Domain specific extensions to enable model adjustability

Model developers would be better supported if they could instantiate their simulation models using domain specific model constructs, for example a doctor with his skills and specific statistics rather than a default resource. The model developer would even be better supported with a complete set of model constructs for his/her domain. This would be a dedicated set of model constructs for problem domains with model constructs only for sub-systems of the specific domain. Examples of sets of model constructs specifically designed for a domain are model constructs to simulate train networks that implement domain specific elements such as rails and stations and model constructs to simulate ship movements in harbors using model constructs to represent water canals and locks (Pater and Teunisse, 1997).

The sets of specific model constructs that Pater and Teunisse (1997) refer to are extensions of a generic simulation environment. In this research an extension of a generic simulation environment for a specific domain is called a "**domain specific extension of a simulation environment**" abbreviated to "**domain specific extension**". A domain specific extension restricts model developers to implementing simulation models of a specific domain, based on a conceptual model of that domain. The domain specific extension consists of model constructs that can be directly derived from the elements of the conceptual model. A model developer no longer needs to make a translation from the conceptual elements to the model constructs and the model developer also does not have to compose several model constructs to represent one system element.

The domain specific model constructs enable the model developer to instantiate the simulation model more easily. The parameterization of the model constructs enables the model developer to make changes to the simulation model to represent solution systems. A domain specific extension reduces the freedom of the model developer, but the advanced model

constructs also reduce the complexity of the instantiation of a simulation model. Adhering to Kasputis and Ng (2000), it is believed that these reductions enable less advanced model developers to make valid simulation models of the problem system in a shorter time span.

Problem owners gather insight by analyzing the outcome of simulation experiments, viewing animation or evaluating input parameters of model constructs. The focus of the research presented here is the effect of domain specific extensions for simulation environments in a simulation study. If problem owners gather more insight into the problem and solution systems without extending the duration of a simulation study, then the simulation study is more effective. Faster development of a valid simulation model that is easily adjusted to support for analysis of solution systems will increase the insight a problem owner gains during the simulation study. The problem owner will better understand the outcome of the simulation model and more solution systems will be analyzed. As a result the effectiveness of a simulation study performed using a domain specific extension will be higher than the effectiveness of a similar study performed using a generic simulation environment.

Domain specific extensions seem to be the best way of performing a simulation study and of providing effective support to problem owners. Nevertheless, a lot of model developers prefer to use generic model constructs, instead of domain specific model constructs in the same simulation environment. One remark often made is that domain specific model constructs can only be applied in limited situations, because the model constructs limit the flexibility of the model developer (Sol, 1982; Page and Opper, 1999; Kasputis and Ng, 2000; Barton et al, 2003; Diamond et al, 2003).

1.7 Research questions and approach

This research is based on the assumption that simulation studies where the challenges defined by Pitt and Robinson have been overcome are more effective than traditional simulation studies, and that the use of domain specific extensions in a simulation study will help to overcome these challenges. The larger effectiveness of the simulation study using the domain specific extensions is expected to result from better insight into the behavior of the system and possible solutions. This insight will be mainly gathered by analyzing the behavior and outcome of different simulation models.

The central research question is therefore:

> *How can domain specific extensions for a simulation environment improve the effectiveness of simulation studies?*

Simulation studies are performed in all kinds of organizations to answer a wide variety of questions. Often the success of a simulation study depends on the political situation within an organization and the priority the project receives from the involved stakeholders (Robinson and Pidd, 1998). The focus of this research is not the success of simulation studies and the results of a simulation study, for example dollars saved or production process improvements. The focus of this research is how to enable model developers to better support problem owners with a simulation study that uses domain specific extension for simulation environments. Hereby the three causes of Robinson and Pidd (1998) are leading to generic attention areas: handle unlimited modeling freedom, model developers need to cover multiple expertises and language mismatch needs to be resolved.

The idea of domain specific extensions is not totally new and it has been applied in several studies. However, the reported low percentage of success has not convinced the community of simulation model developers to adapt the approach of domain specific extensions to its current best practice. The best practice consists mainly of some technical features in the common of the shelf simulation environments. The model developers require more support to ensure they can execute a simulation study using domain specific extensions. The development of these extensions requires the multidisciplinary of the model developer to the extreme to ensure a usable extension for a simulation environment is available. Therefore this research aims to deliver support for the development of domain specific extensions for simulation environments that can be applied for successful simulation studies.

The use of the solution in development of the domain specific extensions should enable model developers to perform their simulation studies in the environment they are accustomed to, it should make the process steps they have to perform simpler and it should enable the model developers to extend the use of simulation models in a specific domain. The verification of these possible increased effects of the use of domain specific extension for simulation environments results in research questions around the ability to define a generic solution that is applicable in different common of the shelf simulation environments, ability to improve the process of performing a simulation study and the ability to reuse the domain specific extension beyond the initial simulation study in a domain.

Research question 2A relates to the difficulties to handle the unlimited freedom in modeling by model developer:

> *What constructs and design approach will enable that domain specific extensions can be defined independent of the generic simulation environment in such a way that the model developer is supported, but not limited to one way of representing a system element?*

Research question 2B relates to the challenge that model developers need to be experts in multiple areas:

> *What methodologies, approaches and techniques can be offered to a model developer to support the use of domain specific extensions in the activities of a simulation study?*

Research question 2C relates to the challenge that model developers do not speak the language of the problem owner:

> *How can be ensured that the domain specific extension gets the model developer closer to the language of the problem owner?*

The starting point for answering the research questions will be the current state of expertise in the field of discrete event simulation. This will be a mixture of the latest state of simulation research as published at conferences and what commercial parties offer in simulation environments and in their consultancy best practices. We follow Cresswell (2003) who suggests to use case studies as these are research instruments to "*explore in depth a program, an event, an activity, a process, or one or more individuals*" (p15). Figure 1.9 demonstrates how this starting point for our research and the use of case studies reflects within the inductive research approach described by Sol (1982).

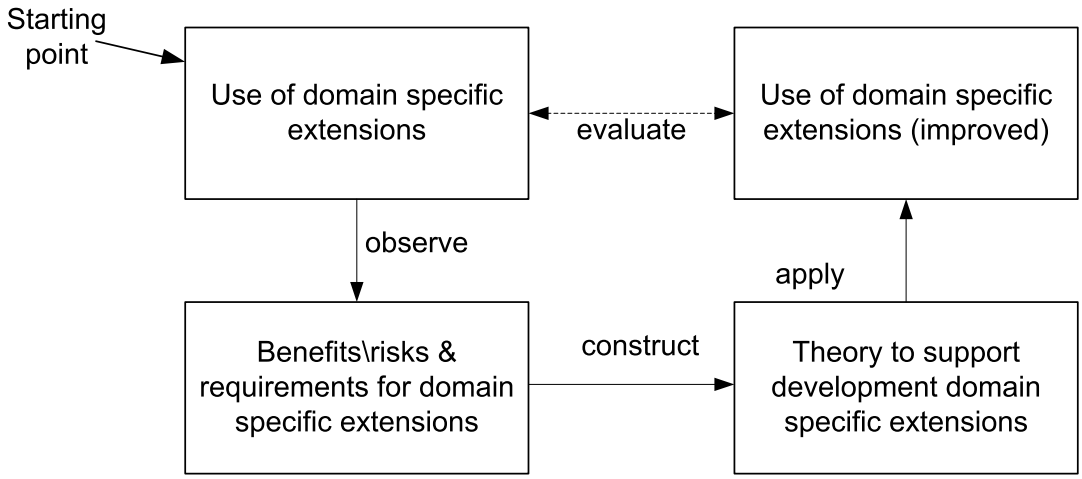


Figure 1.9: Research approach following Sol (1982)

The next step is that we apply the knowledge in case studies with real problem owners for qualitative analysis and set up a laboratory experiment with a fictive simulation study for a quantitative analysis as part of "Use of domain specific extensions" in Figure 1.9. The combination of the two sets of studies will cover the disadvantages that both types of studies carry (Yin, 2003; Cresswell 2003). The disadvantage of the qualitative analysis is that comparison of the simulation study in a traditional way is not possible. The disadvantage of the laboratory settings will be that no real problem owner can

participate and thus the difficulty of communicating, interpretation and resolving the conflicts can not be part of the observations.

The observations of the simulation studies and the quantitative analysis will be used to refine theory. This refining will be done in several steps, initially by translating the observations to benefits and risks for domain specific extensions, followed by the definition of requirements for domain specific extensions that will enable mitigating the risks and enlarging the observed benefits. With all gathered knowledge and experiences we will then construct a theory to support development for domain specific extensions.

We follow Yin (2003) who introduces case study as 1) the research element to cover empirical inquiries in their environment when the contextual conditions cannot be deliberately divorced from the research topic and 2) the case study will deliver more than data points, but will be used to refine the theory and thus the use of case studies is a comprehensive research strategy. The case studies will all cover the global research question, and all individual focus on one of the three sub questions: handle unlimited freedom; support use of domain specific extensions; bridge the gap with the language of the problem owner. In that way we can evaluate whether the theory that we define answers the sub research questions and achieves the overall objective to make simulation studies more effective.

1.8 Outline of the research

The research starts with an analysis of existing domain specific extensions for simulation environments (chapter 2). The knowledge gathered from this analysis will be applied as basis for participative case studies (Yin, 2003) in which domain specific extensions will be developed and used to support problem owners, in their specific domains, and with their specific processes of problem solving (chapter 3). These case studies are expected to confirm the encountered pitfalls and perceived disadvantage of using a domain specific extension that limit model developers in developing simulation models.

Secondly, several laboratory experiments will be performed to compare the use of a generic simulation environment with a domain specific extension (chapter 4). It is expected that these experiments will show that a domain specific extension has advantages over a generic simulation environment. It is also expected that using domain specific extensions in this manner will highlight the disadvantages that model developers encounter when using a domain specific extension for the first time. Observation and surveys of participants in the laboratory experiments and the case studies will be used to provide an overview of advantages that can be achieved and of disadvantages that restrict and prevent model developers from achieving all the possible benefits of using a domain specific extension. The outcome of the laboratory experiment and case studies will be used to define a new concept

and guidelines for domain specific extensions for simulation environments (chapter 5).

This new concept and guidelines will be applied to develop new domain specific extensions and these environments will be used in simulation studies in different domains. All three simulation studies described in chapter 6, 7 and 8 will be used to show the applicability of the concepts and guidelines of domain specific extensions for simulation environments. In addition each of the simulation studies focuses on one of the research questions of research challenge 2. In chapter 6 it will be evaluated whether several simulation environments can be used for the same domain specific extension for supply chains. In chapter 7 a management game of a container terminal design will be supported by automatic tools to verify whether the model developers can be supported in the simulation study process and in chapter 8 the same domain specific extension for simulation environments will be applied to a wide range of simulation studies at Nestlé production facilities.

The findings of the simulation studies in chapter 6, 7 and 8 are combined in chapter 9 to identify whether the solution for domain specific extensions for simulation environments is feasible. Specifically this chapter will provide feedback to the requirements for a solution identified in the beginning of chapter 5. This will lead to answer of the research question and research challenges.

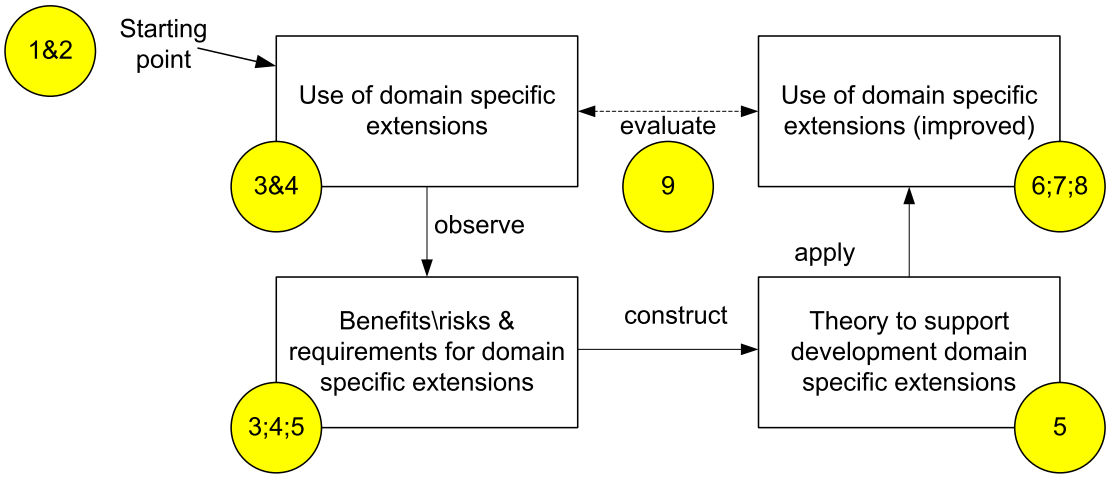


Figure 1.10: Outline of thesis

2 Domain specific extensions of simulation environments

2.1 Domain Specific Extensions; definitions and terminology

Domain specific extensions of simulation environments have existed for several years (Pater and Teunisse, 1997; Baker, 1997). A domain specific extension consists of model constructs that represent a system element of the targeted domain. **Model constructs** are elements in a simulation environment or simulation language that represent a part of the system, and that can be instantiated and parameterized in a simulation model for specific use. Model constructs are **domain specific** if they are a member of a set that is meant to build simulation models for a specific problem domain. The choice whether a set of model constructs is indeed specific for a certain domain is quite arbitrary and can ultimately only be decided by the simulation model developer. Thus a set of model constructs can be domain specific to one person, while another model developer in the same domain might not be able to instantiate certain elements based on the set of model constructs successfully for a simulation study in that domain.

A simulation model will be instantiated according to the system abstractions as defined in the conceptual models. Shannon (1975) defined several activities as part of a simulation study, e.g. conduct experiments, understand the current system, and evaluate strategies for alternative systems. These activities are carried out using a simulation environment. A **simulation environment** is a (set of) computer application(s) that enables modellers to specify a simulation model and conduct experiments with the simulation model. A simulation environment uses a certain simulation language and a simulation formalism to enable the modeler to instantiate the simulation model. The notion of a **simulation formalism** points to the formal meta-model in which a broad class of models can be described (Zeigler et al 2000; Vangheluwe and De Lara, 2002). Popular formalisms are DESS (Differential Equation System Specification, Zeigler et al 2000) for continuous modeling and DEVS (Discrete Event System Specification, Zeigler et al 2000) for discrete-event modeling. Formalisms that build on DESS are e.g., differential equations, System Dynamics (Forrester, 1999), and Bond Graphs (Cellier, 1992). Extensions of DEVS are e.g., the process interaction formalism (Nance, 1993) and the event scheduling formalism. In simulation environments, these meta-models are made more specific by providing a **simulation language** that builds on the simulation formalism, and that provides a set of model constructs to the modeler. Many of these simulation languages are programming languages, early examples are SIMULA (Dahl and Nygaard, 1966) with DEMOS (Birtwistle, 1979), GPSS (Schriber, 1974),

SIMAN (Pegden, 1990), and Simscript II (Kiviat, 1966). A good overview is provided in Nance (1993). With the increased power of computers and graphics, and to avoid programming, graphical modeling environments were created for simulation on top of the simulation languages. These are also referred to as drag-and-drop environments or grab-and-glue environments (Paul, 2002; Eldabi et al., 2003). Many of the popular general-purpose simulation environments are of this type. Swain (2007) lists over 50 commercial simulation environments. Rockwell Automation (2007) provides an overview of the most mentioned commercial simulation environments at the WSC conference of 2006. These include the simulation environments that will be discussed in this thesis: Arena (Kelton, Sadowski, and Sadowski, 2002; Bapat and Sturrock, 2003), ProModel (Harrell and Price, 2003), AutoMod (Rohrer, 2003), Extend (Krahl, 2003), SIMUL8 (Haige and Paige, 2004), eM-Plant and its predecessor Simple++ (Kalasky and Levasseur, 1997), and Enterprise Dynamics (Britals, 2008). Even the Java-based AnyLogic simulation environment (XJ Technologies, 2005) uses the drag-and-drop metaphor to enable users to create their simulation models. The library of components in these general purpose simulation environments from which the simulation model is assembled contains the model constructs. We refer to these basic model constructs as “generic”, because the developers of the simulation environments aim at generic use so that all kinds of systems can be represented with the model constructs. Almost all simulation environments allow users to develop extensions to their generic model constructs (Valentin and Verbraeck, 2007). In this context we define an **extension** as a coherent set of model constructs aimed to represent a system or systems in a particular domain.

How a simulation study is carried out was shown in figure 1.8. The first steps are to define the conceptual model, select the model constructs and to instantiate the simulation model in the chosen simulation environment. **Conceptual modeling** involves **abstraction** of the system into generic classes of elements instead of a full listing of the element instances. As Van Gigch (1991, p. 19) states it: “In the usual sense to abstract means to isolate certain characteristics from others. It also refers to an action of the mind, a mode of inquiry which seeks to generalize (i.e., to consider lower-level statements from a metalevel perspective and to extract their common features).” In the process of defining the conceptual model, the model developer decomposes the system into system elements. **Decomposition of a system** is to separate the system into smaller elements that contain a coherent part of the functionalities of the system with their relations. As Sage and Armstrong (2000, p.7) state it: “Because large-scale systems are inherently complex in the sense of being comprised of many subsystems, systems often can be better understood by organizing their parts into groups based on function or some other organizing principle. Often systems are

organized into hierarchies". The first step is to decompose the system into system elements. The second step is a translation of the system elements to the model constructs of the simulation environment, i.e. selecting the suitable model constructs to represent each individual system element and their relations. The final step is to compose the simulation model from the selected model constructs. **Composition of a simulation model** is to combine several model constructs to represent the system elements and, ultimately, the entire system under consideration. The composition of the model constructs will also enforce the relations between the system elements to be included in the simulation model. In most simulation environments, this is the "glue" part of the grab-and-glue approach (Eldabi et al 2003; Eldabi et al 2004). The decomposition and composition steps to develop a simulation model are shown in Figure 2.1. This figure also shows that the simulation model is composed from a selection of the available model constructs. A limited number of the model constructs that are part of the simulation environment is often sufficient to compose the simulation model to represent the system.

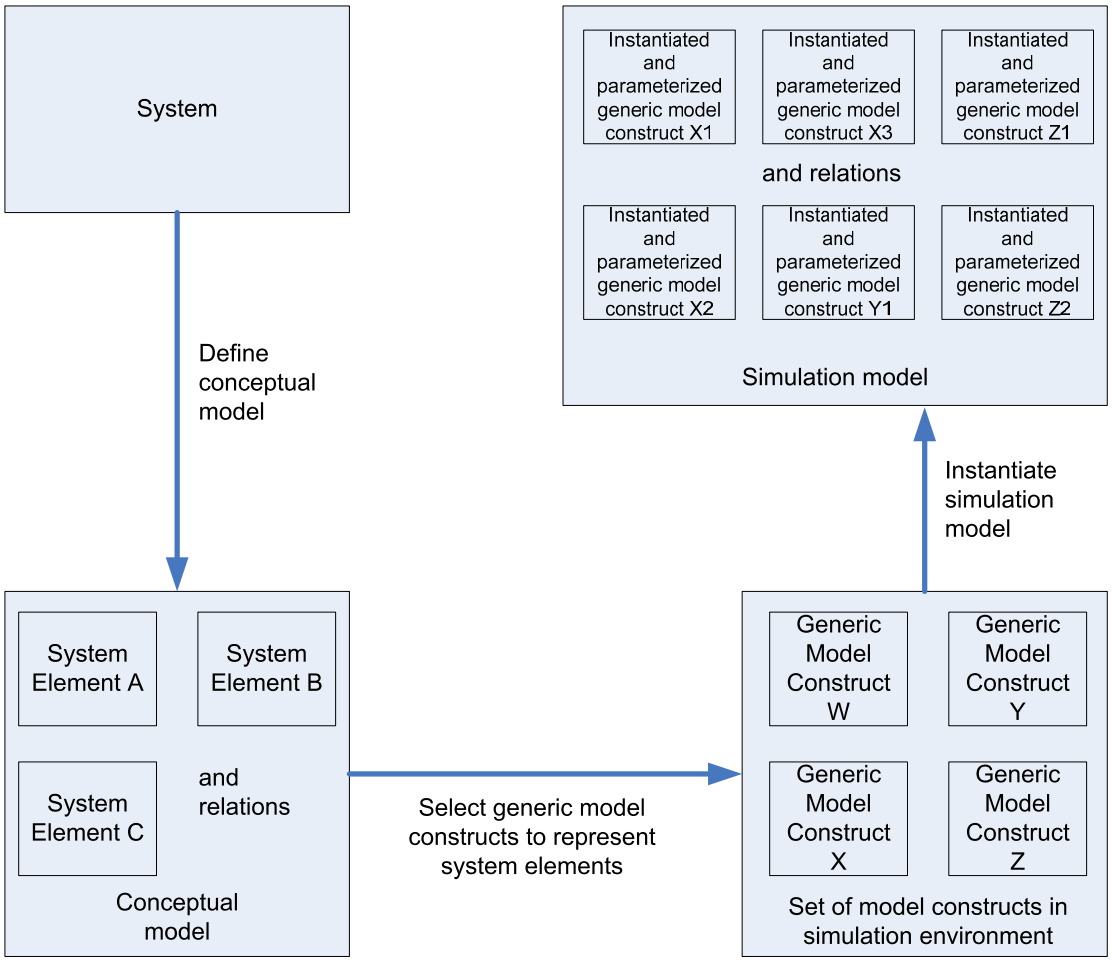


Figure 2.1: Decomposition of a system and composition of a simulation model

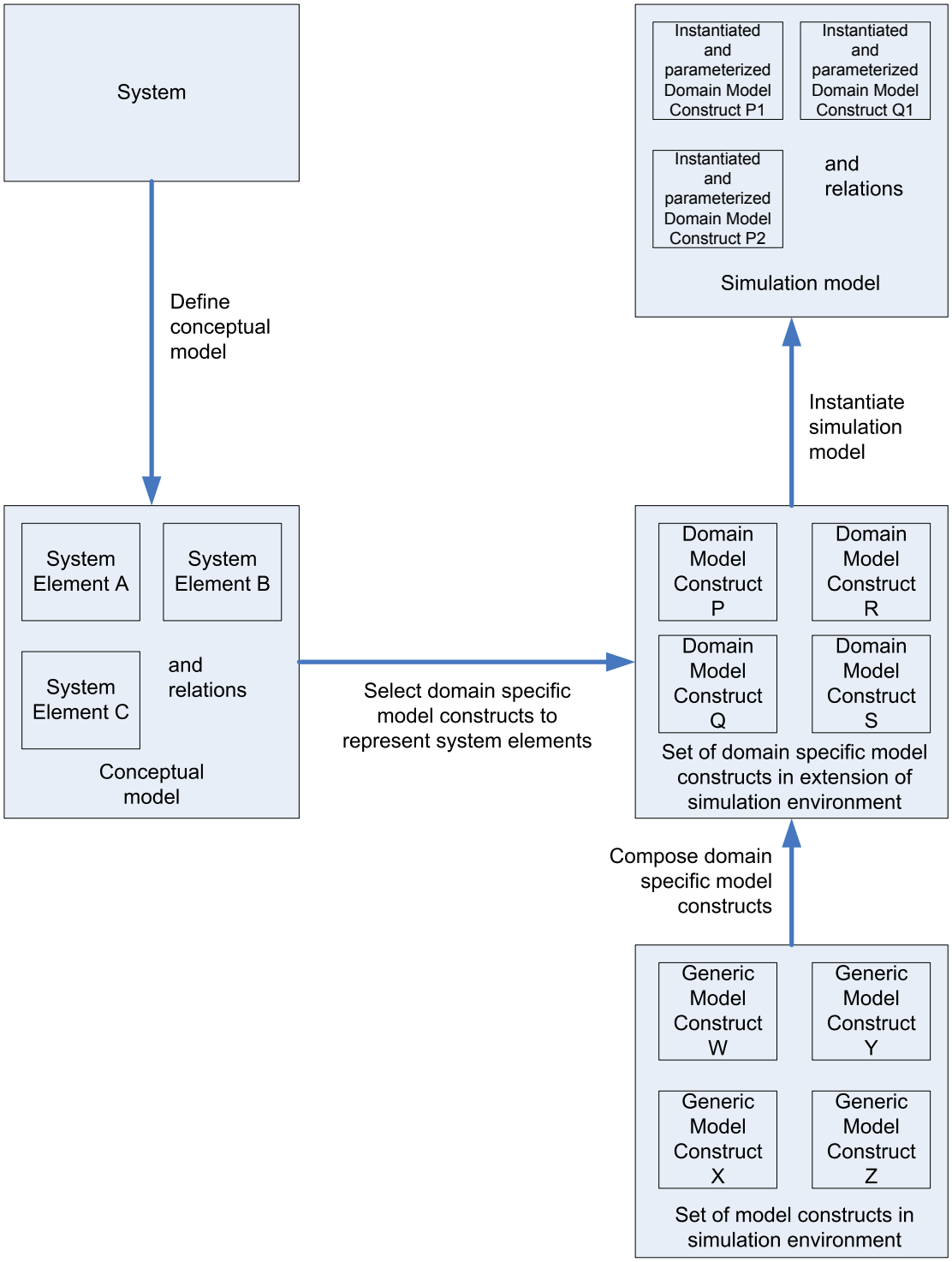


Figure 2.2: Decomposition of a system and composition of a simulation model using domain specific model constructs

Domain specific model constructs are developed by assembling one or more model constructs of the generic simulation environment. Each model construct of a domain specific extension is based on a composition of model constructs of the generic simulation environment. The selection of model constructs to represent the system elements is now a different process, because the set of model constructs is different. Figure 2.2 shows how a

model developer can now select directly from the domain specific model constructs. Figure 2.3 and Figure 2.4 show the different ways of representing a system element in a simulation model using the example from the previous chapter.

There are several other differences between the generic and domain specific model constructs besides the hidden complexity. In Figure 2.3 the low-level process flow is shown in terms of the generic model environment (create, process, dispose), not in terms of something a problem owner would understand. In Figure 2.4 an icon is used of an operator with a tool as a representation, which can be much easier understood by a stakeholder in the problem domain. This visualization enables the model developer to assess the structure of the simulation model more quickly, and the problem owner will directly recognize the system element.

Another important difference between the model with the generic and domain specific model construct is the ability to make changes to the representation of the system element. When using generic model constructs, a different parameterization of the system element, for example a shorter processing time, might require changes in several of the generic model constructs that are used to represent the system element. The model developer also needs to know in which of the model constructs to make the change for the correct parameterization. When using domain specific model constructs, however, parameters are displayed only once and recognizable (non simulation-specific) terms can be used. Finally, the generic model constructs have parameters that are not applicable for the representation of the system element. All these additional parameters might confuse model developers and complicate the process of instantiating the simulation model. In the model constructs of a domain specific extension, only those parameters are shown that are expected to be changed by the modeler.

The system of the sealing department, introduced in the previous chapter, includes the handling of breakdowns of the sealing machines. Figure 2.3 and Figure 2.4 are both a part of the simulation model representing the system element “Breakdown process”. In Figure 2.3 the system element is instantiated using three different model constructs of the generic simulation environment Arena. In Figure 2.4 the same system element is instantiated using one domain specific model construct of a domain specific extension for modelling of factories.

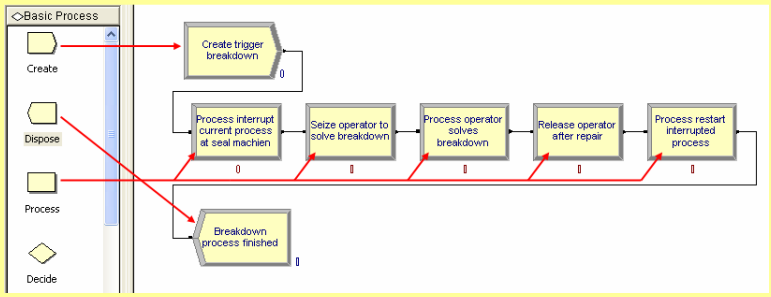


Figure 2.3: Implementation of system element “Breakdown process” using generic model constructs “create”, “process” and “dispose”

The domain specific extension for computer factories extends the generic simulation environment Arena. The model construct “Breakdown process” uses, internally, the same generic model constructs as shown in Figure 2.3. The generic model constructs shown in the circle in Figure 2.4 are used to compose the domain specific model construct “Breakdown process” by the developer of the domain specific extension “Computer Factory”. In this case, the two ways of representing the system element will thus result in exactly the same output of the simulation model.

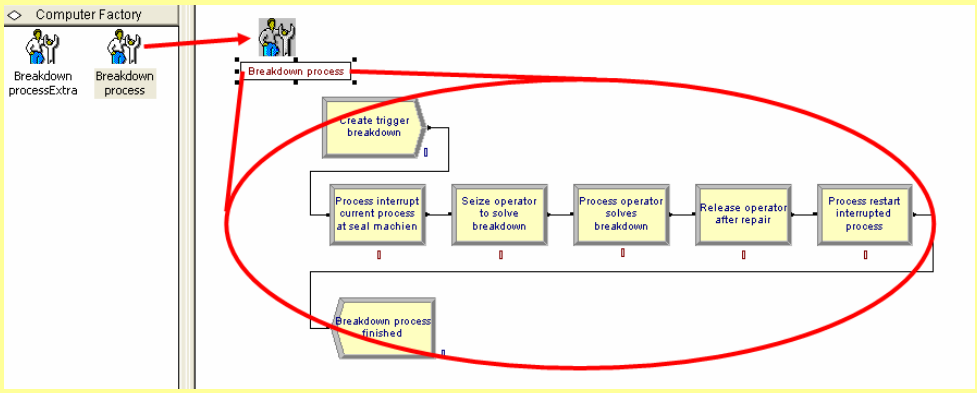


Figure 2.4: Implementation of system element “Breakdown process” using one domain specific model construct, which automatically instantiates all required generic model constructs

2.2 Representation of system elements in a domain specific extension

2.2.1 Decomposition, abstraction, and generalization

The model constructs of a domain specific extension are representations of system elements in a domain. In order to make the extensions applicable in multiple simulation studies, the system elements that are represented in a domain specific extension should not be the result of decomposing and abstracting only one system. In a simulation study that uses a generic simulation environment, the particular system that is studied is **decomposed**, identifying individual instances of parts in the system (see top part of Figure 2.5). In Figure 2.5, a further **abstraction** results in the following set of system elements: hexagon, cross, ellipse, square and circle (bottom of Figure 2.5).

The set of system elements identified in different systems within the same domain can vary. In Figure 2.6 (top), the decomposition of three systems results in 15 different system elements. A closer look at these 15 system elements shows us that they are not completely different. For example, the decomposition resulted in three system elements that appear to be triangles. The differences between these three systems elements, in addition to size and position that had already been turned into properties, is the rotation. The system elements with commonalities, e.g. the three triangles, can be further generalized to one system element that can be configured via parameters. In abstraction and generalization commonalities are identified in different system

elements to allow these different system elements to be represented by one system element. We reserve the term **abstraction** to the process of identifying the system elements based on a set of instances. In object orientation, this process is equivalent to defining object classes. We use the term **generalization** to the process of reducing the set of system elements further by further parameterization of system element properties. This is equivalent to defining superclasses in object orientation and applying the inheritance relation. The result of the generalization process depends on the trade-offs made by the modeler between many system elements with a few parameters versus few system elements with many parameters.

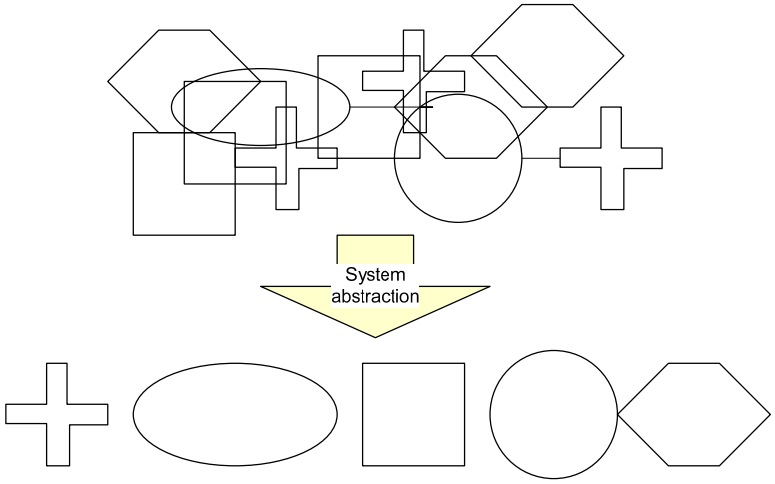


Figure 2.5: System decomposition and abstraction of a system

A possible generalization of the 15 system elements of the decomposition of three systems in Figure 2.6 is the depicted set of 7 system elements. By adding a parameter “number of vertexes”, the hexagon, square and triangle could be further generalized into a system element called polygon. Taking this to the extreme, one system element called shape, with a large number of parameters, could be sufficient to model all three systems of Figure 2.6.

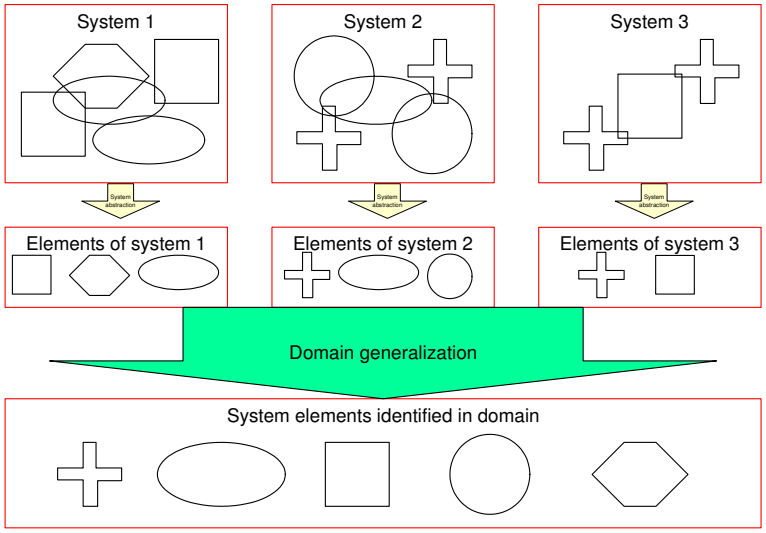


Figure 2.6: Domain generalization

2.2.2 System abstraction

Simulation studies require a level of abstraction of the system that depends on the goal of the simulation study. In a factory problem, for instance, abstraction can be applied to the processes and the equipment, see chapter 1. Abstraction of the equipment is performed by identifying classes of equipment, and generalizing into more common classes where appropriate, thus obtaining fewer system elements or more common system elements. Abstraction of the processes is performed by leaving out process rules and process details and thus obtaining combined system elements representing process steps.

The decomposition into the types of system elements is often done using two different views of a system. The first view for decomposition is that comprising equipment and infrastructure or objects. We follow Pollacia and Delcambre (1997) and we will refer to this decomposition view as "**object oriented decomposition**". The second view for decomposition is that comprising services or activities or processes. We will refer to this decomposition view as "**process oriented decomposition**".

An example of a domain specific extension that matches the two systems views for decomposition is "Contact Center" based on the generic simulation environment Arena (Bapat and Sturrock, 2003). The domain specific extension Contact Center aims at the modelling of call centers. The system elements identified using the object oriented decomposition are "agent" and "telephone line". The system elements originating from the process oriented decomposition are processes like "handling a call by an agent", "routing decision" and "handling queue priorities". The two types of model constructs enable the model developer to experiment with changes to the processes and to the objects.

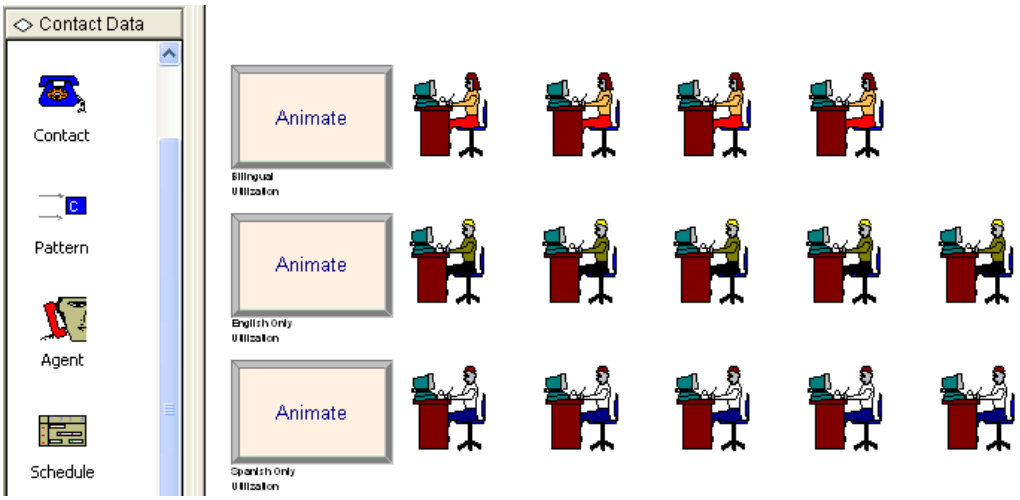


Figure 2.7: Call center agents with different characteristics for handling calls

Figure 2.7 and Figure 2.8 are two screen dumps of simulation models instantiated with the domain specific extension Contact Center. Figure 2.7

shows the operators handling incoming calls. All operators on the right hand side of the screen dump are instances of the abstracted system element “agent”. Each of the agents is further characterized by the setting of certain parameters. Figure 2.8 shows the process flow of handling calls in a call center depending on the origin and call attributes. Further detail could be achieved by dividing one or more of the flows into sub flows to make additional distinction between calls in the call center.

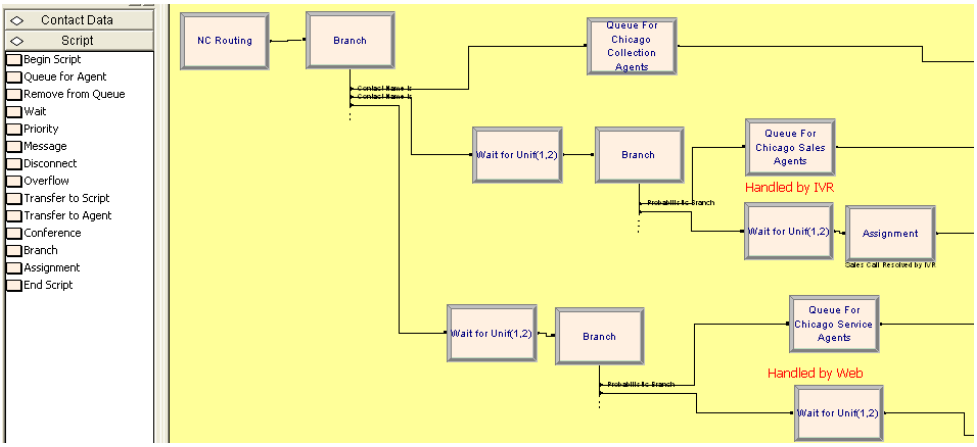


Figure 2.8: Process steps for handling calls

2.3 Design process for domain specific extensions

Figure 1.8 in chapter 1 showed the generic process of a simulation study. How to perform simulation studies was described as the first generic simulation environments became available (Birtwistle, 1979; Shannon, 1975). There is, however, much less literature available on how to carry out the process of developing domain specific extensions. In this section we give some guidelines based on the available literature. These guidelines will be used to develop domain specific extensions for the case studies (chapter 3) and laboratory experiments (chapter 4).

2.3.1 Generic simulation environments

Most generic simulation environments have features to develop domain specific model constructs. They provide ways of combining a set of simulation model constructs into an advanced model construct. Two examples are shown in Figure 2.9 and Figure 2.10 for the generic simulation environments Witness and Enterprise Dynamics. These examples show that the development of a model construct is done with just one mouse click. Other examples are the generic simulation environment Arena, which allows for the development of advanced custom model constructs (Rockwell Software, 2000), and eM-Plant, in which software is built in a hierarchical manner using so-called frames (Kalasky and Levasseur, 1997). Examples of the use of both these simulation environments will be given in subsequent chapters.

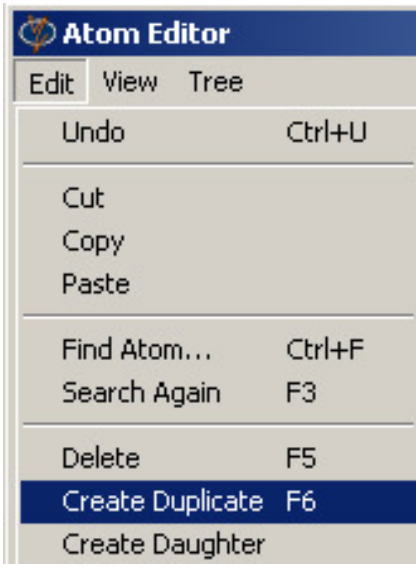


Figure 2.9: Menu for duplicating existing model constructs in Enterprise Dynamics

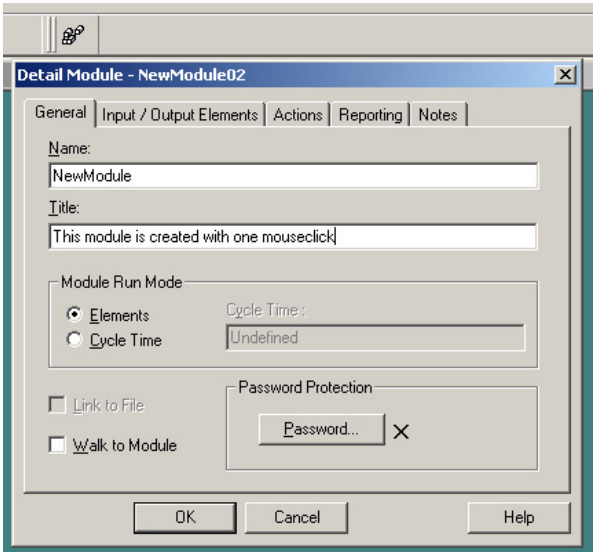


Figure 2.10: User-interface to develop a model construct out of a part of a simulation model in Witness

In our opinion the aim of a domain specific extension for a simulation environment is to ease simulation model development of different systems, by providing more than the ability to combine and reuse model constructs of previous simulation studies carried out in a domain. A **domain specific** extension is a set of system elements to be simulated in a particular domain. In that sense, its potential application base is less than that offered by generic simulation environments. Although the “technical” way a model construct is developed is important, we should also look at the process by which model constructs can be developed by model developers to make a simulation model in a domain. Manuals of generic simulation environments do not address this issue; they focus – understandably – on the technical issues.

2.3.2 Process descriptions in literature

Research in the field of object oriented simulation models has paid attention to the use of model constructs to assemble a simulation model. Several researchers have built on the use of object orientation in software engineering and its suitability for simulation. Jacobs (2005) and Tyszer (1999) applied object orientation, but they mainly focus on its use to develop a new simulation environment, not to represent systems in a specific domain. An object oriented simulation method has been described by Hill (1996). He evaluated software engineering approaches and adjusted and combined them into one approach for defining object oriented simulation models. He suggests an approach in three phases which he calls M2PO. In the first phase object classes are identified and, for all classes, the dynamics and object life cycle are defined. In the second phase, system specific elements are added and in

the third phase the collaboration between objects is included. After the third phase the objects can be implemented in an object oriented simulation environment.

The approach described by Hill (1996) focuses on object oriented models for only one system. He considers the need to adjust simulation models to carry out additional experiments, but his approach is not aimed at designing simulation objects for domain wide applicability. Therefore the approach that he describes does not fit in its current version as a prescription for the development of a domain specific extension, because the model constructs of these environments need to be flexibly applicable to cover the wide range of demands for more experiments that problem owners demonstrate.

Zobrist and Leonard (1997) give several descriptions of object oriented simulation software, frameworks and methods. In that book, Kim and Ang (1997) present a framework, which builds on DEVS (Zeigler, 2000), in which they apply five principles: 1. Functions may be reused in the development of models (function abstraction); 2. Data may be reused in the development of models (data abstraction); 3. Models may be reused in the construction of composite models which in turn are reused as components of higher level composite models (composition); 4. Models may be reused in the development of new models that are slightly different from old ones (inheritance); 5. Models may be reused in a variety of applications (use of libraries). We will see these principles back in later chapters.

Pater and Teunisse (1997) have developed a domain specific extension for cargo rail networks. In their article they generalize the approach they used to design a domain specific extension. The first step of the process they described is an analysis of requirements of the problem owners for the type of information they are interested in, by identifying the system elements using object oriented decomposition in their domain. In this case the authors abstracted the system objects on a very high level, leading to constructs such as a “pipe” and a “node”. Secondly, they applied process oriented decomposition to define processes to be part of their simulation models. In their case of rail cargo transport, they identified e.g., safety mechanisms and traffic control requirements. All model constructs are at quite a high level of abstraction. The implemented model constructs of the domain specific extension are then extended with additional objects and processes whenever these are required in later studies. Pater and Teunisse (1997) refer to their process of abstraction, generalization and decomposition as a top-down approach.

2.3.3 Developing domain specific extensions

We combine the top-down approach of Pater and Teunisse (1997), the notions of Kim and Ang (1997), and the object oriented approach of Hill

(1996). We can outline an approach for the development of domain specific extensions. In this approach we will decompose a system in an object oriented and a process oriented manner. The steps of this approach for development of domain specific extensions are shown in Figure 2.11. The approach is a first draft and therefore the steps are suggestions and not prescriptive.

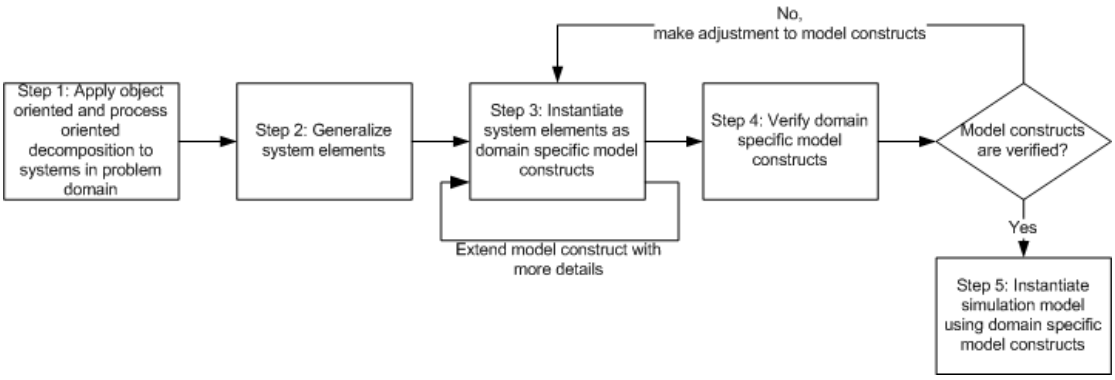


Figure 2.11: Developing a domain specific extension

Step 1. Object oriented and process oriented decomposition and abstraction. The development of a domain specific extension should take into account observations of not just one individual system, but preferably several systems, to get a more complete picture of the domain. The decomposition and abstraction should be carried out for both the objects and the processes that can be identified in these systems. Pollacia and Delcambre (1997) call this object flow modeling.

In this first step of the development of a set of model constructs, the problem domain should be decomposed to identify those system elements that could be turned into model constructs later. Object oriented decomposition and abstraction will result in a set of system elements. Hill (1996) uses UML class diagrams for his object-oriented decomposition and abstraction. The description of each object class should include attributes and behavior, because this will enable model developers to understand what is included in the model constructs that will be based on the decomposed system element.

Process oriented decomposition and abstraction will result in process descriptions that can be derived from informal system descriptions. The result will be small process descriptions relating to one or more system elements identified with the object oriented decomposition. The overall result of the decomposition and abstraction will be a set of system elements.

Step 2. Generalize system elements. Generalization should be applied to the identified system elements to reduce the number of system elements and reach a manageable and understandable number of model constructs. The

definition of these model constructs should be extended with parameters to enable modelling of different systems using the model constructs that represent different system elements as suggested by Kim and Ang (1997).

Step 3. Instantiate system elements as domain specific model constructs. The generalized system elements should be translated into a domain specific model construct in a certain simulation environment or simulation language. Each domain specific model construct is a composition of generic model constructs, which represents a system element and uses parameterization.

The top-down approach suggested by Pater and Teunisse (1997) is applied by first instantiating model constructs as abstract elements. Once a first version of all model constructs is implemented, the model constructs can be extended with details. The process of adding the details is represented in Figure 2.11 by the arrow “Extend model constructs with more detail”. The details could include additional functionalities, but also an improved user-interface, visualization of the state of the model construct, or performance indicators.

Step 4. Verify domain specific model constructs. Tests should be performed to make sure that the created model constructs are behaving as the developers expect, i.e. verification. These tests are performed by modeling one or more systems that allow a model construct to be tested, and studying the input-output behavior of the model construct. If the test is not successful, the developers should make adjustments to the model construct. These adjustments can be an alternative combination of underlying generic model constructs, alternative parameter settings, or alternative calculations of performance indicators by the model construct.

Step 5. Instantiate simulation model using domain specific model constructs. The model constructs should be used to implement a simulation model once all model constructs are verified and have the desired level of detail. At this moment the simulation study can be performed as described in Figure 1.8.

2.4 Advantages of using a domain specific extension

Using a domain specific extension in a simulation study does not necessarily change the activities in a simulation study. We hypothesized in chapter 1 that a simulation study using domain specific extensions will be more effective than one without. The improvement in effectiveness is expected to take place in all of the activities of a simulation study, as will be discussed below.

Activity 1. Problem description & define conceptual model: the activity of conceptualization of the original system is inevitable, but the availability of a

domain specific extension will facilitate the conceptualization. The conceptualization does not need to be done from scratch, but can start from observing which of the system elements of the domain specific extensions are applicable to the specific system. In many cases, the object classes and process descriptions that have been used to create the domain specific extension can be reused to conceptualize the system at hand.

Activity 2. Select model constructs: as the model constructs of the domain specific extension have a clear relationship with the system elements identified in the problem domain, the conceptual model is structured according to the model constructs of the domain specific extension. Figure 2.1 shows that in traditional simulations, a conceptual model is translated from the system elements via instances of generic model constructs to the simulation model. Figure 2.2 shows that the translation using a domain specific extension does not involve generic model constructs, but only the model constructs of the domain specific extension.

Activity 3. Data collection: this activity is not shown in Figure 1.8, because the description is focused at the steps of the model developer. Data collection is easier for domain specific model constructs than for traditional simulation modeling activities. The data to be collected is determined by the parameters of the domain specific model constructs, where the model constructs are nicely mapped onto system elements. This reduces discussion and confusion regarding the type and format of the data needed. Furthermore, data gathering can begin directly at the start of a project, and does not have to be postponed until insight is gathered about the type of data that is required.

Activity 4. Instantiate simulation model for original system: fewer model constructs need to be used, thanks to the hiding of complexity (Kasputis and Ng, 2000; Altiok, 2001). Traditionally, model coding of a system element normally involves many model constructs of a generic simulation environment, but each system element can now be modelled by one model construct of the domain specific extension. An example is shown in Figure 2.3 and Figure 2.4.

Activity 5. Verify and validate simulation model for original system: detailed testing of complex logic is needed in traditional simulation modeling. When we assume that the developer of a model construct of a domain specific extension has performed sufficient testing before handing over it to the model developer (step 4 in the method), a model developer does not need to test the model construct of a domain specific extension anymore in detail. This is comparable to a model developer instantiating a model construct in a generic simulation environment, where the developer also does not test whether that construct works correctly (Baker, 1997).

Activity 6. Analyze output of simulation model: the output of domain specific model constructs is usually standardized. Therefore, in each simulation model using the constructs, the same type of data will be produced, thanks to the definition of performance indicators in the model constructs. Often, problem owners in a domain are interested in the same type of performance indicators, even though they are part of different systems.

Activity 7. Define solution for analyzed output: identifying possible solutions based on the output will be similar to a normal study, but an additional source for new solutions will be provided by the parameterization of model constructs. The system element representation of generic model constructs keeps the implementation of possible changes hidden for the modeler, though. The analysts can directly see the possible changes to the parameterization of the domain specific model construct.

Activity 8. Instantiate simulation model for identified solution: the user interface enables model developers to adjust a simulation model more easily and therefore carry out simulation experiments of system alternatives easier (Pater and Teunisse, 1997; Altiok et al, 2001).

Activity 9. Verify and validate simulation model for identified solution: this is faster for the same reasons that apply to the verification of the simulation model for the original system.

Activity 10. Analyze output of simulation model for identified solution: faster for the same reasons mentioned for the project step “analyze output of the simulation model”.

2.5 Risks of using a domain specific extension mentioned in literature

Unfortunately, the use of domain specific extensions has not always resulted in effective simulation studies in practice. Simulation practitioners have mentioned several risks they encountered while using or trying to use model constructs of domain specific extensions within simulation studies, see for example: (Sol, 1982; Balci, 1997; Pater and Teunisse, 1997; Page and Opper, 1999; Davis et al, 2000; Kasputis and Ng, 2000; Banks et al, 2001; Diamond et al, 2002; Barton et al, 2003). We have allocated the risks mentioned in literature to the activities of a simulation study (Figure 1.8). As for activities 3 ‘Data Collection’ no risks were mentioned in literature, the data collection activity is not mentioned in the overview below.

Activity 1. Problem description & define conceptual model: the use of a domain specific extension can limit the scope of the model developer. Model developers tend to consider only the capabilities of the model constructs of the domain specific extension. This can restrict the problem description and therefore not match all the requirements of the problem owner (Sol, 1982).

Activity 2. Select model constructs: according to Balci (1997), model developers have limited trust in model constructs of domain specific extension and thus less trust in the simulation models developed using these model constructs; as a result they will not select domain specific model constructs to compose a simulation model. Kasputis and Ng (2000) describe how a model developer might not have insight into whether a domain specific extension is suitable for representing a particular system and thus decide not to use domain specific model constructs that actually might be suitable. On the other hand, Pater and Teunisse (1997) describe examples of model developers that

overestimated the functionalities provided by a model construct of a domain specific extension and used model constructs that were not suited to represent a system element.

Activity 3. Data collection: No risks are documented regarding data collection when model developers use domain specific extensions.

Activity 4. Instantiate simulation model for original system: Barton et al (2003) indicate that model developers do not always understand the model constructs of domain specific extensions. Model developers do not know how to parameterize the model construct, how to interface the model construct with other model constructs or what the state variables of the model constructs mean in the real system. This results in simulation models that have an incorrect representations of system elements or an ill-defined state of the model construct.

Activity 5. Verify and validate simulation model for original system: verification and validation of a simulation model can take more time when using domain specific model constructs, because the model developers have either instantiated the wrong model constructs or parameterized the model constructs incorrectly. The model constructs are a black box to the model developer, thus identifying what has been done wrong by a model developer is hard to identify (Diamond et al, 2002).

Activity 6. Analyze output of simulation model: the model constructs calculate performance indicators that the developers of the domain specific extension find useful. Problem owners might be interested in more or different performance indicators for a system, ones that are not included in the domain specific extension (Diamond et al, 2002; Barton et al, 2003).

Activity 7. Define solution for analyzed output: the capabilities of the model constructs influence the type of experiments that the model developers will consider. The model developers are restricted by the model constructs in their thinking and their modeling (Sol, 1982; Page and Opper, 1999; Kasputis and Ng, 2000; Barton et al, 2003; Diamond et al, 2003).

Activity 8. Instantiate simulation model for identified solution: when solutions are defined that can not directly be modeled using the model constructs, an adjustment of the model constructs of the domain specific extension might be required. The availability of the developer of the domain specific extension, the structure of the model constructs or the concepts applied in the design of the model constructs might make it difficult or impossible to model these solutions (Davis et al, 2000).

Activity 9. Verify and validate simulation model for identified solution: the same risks apply as during the verification and validation of the simulation model of the original system.

Activity 10. Analyze output of simulation model for identified solution: the same risks apply as during analysis of the output of the simulation model of the original system.

These risks are encountered in situations where a domain specific extension was already available for the system to be simulated. However, only a very small set of domain specific extensions is available and according to Page and Opper (1999) and Barton et al (2003), designing and developing a domain specific extension that can be used by other model developers is a difficult and time consuming investment.

In general, the encountered risks are caused by a lack of understanding of the model developers regarding the usability of (model constructs in) the domain specific extension, and by the limited flexibility of the model constructs. In several cases reported in literature, the simulation practitioners found ways to handle the risks. In general, the fact that these risks occur might lead to simulation studies that do not provide sufficient insight for problem owners. The occurrences of these risks have two causes. First, model developers do not use domain specific extensions and keep carrying out simulation studies in the old way, which is a lost opportunity of capitalizing on the advantages mentioned in section 2.4. Second, model developers that use the domain specific extension cannot produce valid answers for the model developers.

2.6 Conclusion

Although domain specific extensions clearly have added value, model developers are hesitant to use domain specific extensions because of several risks that apply when using them in practice. Providing solutions to overcome these risks will be an important step towards acceptance of domain specific extensions by simulation model developers. Not much literature about domain specific extensions is available, however, and the advantages and risks are still poorly understood. In order to get a better understanding of the advantages and disadvantages of the use of domain specific extensions, we will carry out a number of case studies in the next chapter using a common-of-the-shelf simulation environment that allows for the creation of domain specific extensions. These case studies lead to a rich list of advantages and risks that will form a basis to provide solutions that can deal with the risks.

3 A qualitative analysis in domain specific extensions

3.1 Introduction

It was shown in chapter 2 that model developers observed a number of risks when using domain specific extensions, which prevent them from using such extensions. An analysis of the limited amount of publications on this matter shows that the main causes are a lack of trust in the model constructs of the extensions and perceived difficulties with the maintainability or adjustability of a domain specific extension. Further, model developers are unclear as to whether using domain specific extensions will provide more benefits than using model constructs of generic simulation environments, and therefore, the developers are reluctant to take the time required to get to know a domain specific extension and to use this environment instead of a generic simulation environment with which they are familiar.

We carried out two case studies in which we developed a domain specific extension, to observe whether we would encounter these risks and whether the risks cause more problems than benefits. We will use the theory and concepts described in chapter 2 and use the simulation environment eM-Plant for our case studies to obtain a close match between the object oriented decomposition and the hierarchical object modeling present in eM-Plant.

The domains of the case studies had the potential that more simulation studies could be carried out, which made us decide to invest in the development of domain specific extensions. The first domain for which a domain specific extension was developed and applied concerned the modeling of advanced control techniques for Automatic Guided Vehicles (AGVs). The domain specific extension was used to develop simulation models in a project for underground transportation of cargo around Amsterdam Airport Schiphol, abbreviated to OLS (Ondergronds Logistiek Systeem = Underground Logistic System). Simulation models have been used in this project to evaluate hundreds of different designs for terminals for loading and unloading of the AGVs (Verbraeck et al, 1998b; Verbraeck et al, 1999; Van der Heijden et al, 2002; Versteegt, 2004).

The second domain concerned the modeling of passengers at airports. Simulation models have been developed using a domain specific extension to model movement and activities of passengers in airports. These simulation models have provided support for problem solving in three projects. The studies looked at passenger terminals at Amsterdam Airport Schiphol (NL) and John F. Kennedy in New York, USA at different levels of detail (Blom and Korf, 2000; De Witt-Hamer, 1999; Valentin, 2002).

The simulation studies described in this chapter were all carried out using just a domain specific extension. These systems have not been modeled simultaneously using model constructs of a generic simulation environment. The simulation studies therefore did not result in a comparison between using a domain specific extension and using a generic simulation environment, but rather insight was gathered into the use of domain specific extensions in real, large scale simulation studies that are expected to benefit most from the use of domain specific extensions. The outcome of this chapter should be a confirmation of the expected benefits and encountering of risks. The simulation studies were also expected to provide some insights in how the risks were avoided or mitigated once they were encountered to make simulation studies with domain specific extensions even more effective. Chapter 4 describes a number of laboratory experiments in which the same problem is addressed with generic simulation environments and domain specific extensions, and provides a comparison.

3.2 Exploratory case study 1: OLS design of terminals

3.2.1 Introduction OLS-project

The fast growing traffic on the Dutch roads is seen as a rapidly increasing problem for efficient and on-time transportation of goods. The Dutch government fears that traffic delays will reduce the competitive value of the Netherlands as a logistics and transportation country. As a possible solution it is promoting the use of new transportation technology (CTT, 1997). The transportation of flowers worldwide via the flower auction at Aalsmeer is an example where the Netherlands acts as a transit country. Flowers from all over the world are brought into Aalsmeer by air (or by rail or truck), and, after auctioning, often exported the same day by airplane (or by rail or truck). The transportation process between Amsterdam Airport Schiphol and flower auction of Aalsmeer, which is now done by truck, is vital for the auction, because flowers that miss the airplane have to wait an extra day. These flowers will be worthless and cannot be exported. The risks of delays with transportation by truck are increasing, therefore an alternative transportation mode that will provide higher reliability is necessary.

In 1997 the Dutch government started research into the use of Automatic Guided Vehicles (AGVs) to transport goods, mainly flowers, between the flower auction of Aalsmeer, different terminals at Amsterdam Airport Schiphol and a brand new rail terminal in Hoofddorp. This project should result in a transportation system capable of handling 3.5 million tons of cargo per year in 2020, using a tunnel system, see Figure 3.1. These AGVs should replace truck movements, reducing pressure on traffic and increasing the time window for handling flowers at the auction. This project is called OLS-Schiphol, whereby OLS stands for Underground Logistic System.

Figure 3.1: AGVs moving through tunnels underneath the airport (CTT, 1997)

In the original design of the OLS-Schiphol, the terminals of the OLS-Schiphol were connected by a tunnel system of up to 20 km (CTT, 1997). 200 to 400 AGVs were designed to be used to transport the goods. Figure 3.2 shows the most likely route of the design in 1997 and the main alternative with a dotted line. Deciding which of the many possible different layouts to use was one of the many topics that needed to be dealt with in the OLS project. The transportation system of AGVs should consist of state of the art techniques for vehicles, tunnel constructions, control mechanisms and loading and unloading equipment. Each of these techniques had a wide range of alternatives and each possible decision has effects on the logistic and economic performance of the system in one way or the other. Simulation models were thought of as being able to provide answers to the questions of the system designers regarding the scale of the system and detailed control of the AGVs at small transit terminals.

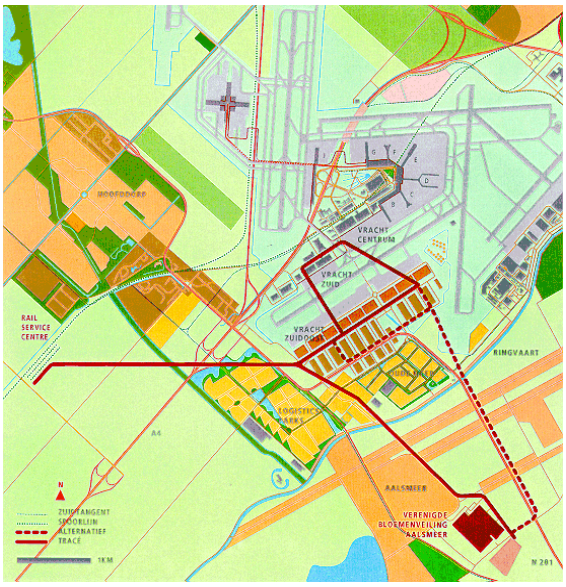


Figure 3.2: Map of the routes between Amsterdam Airport Schiphol, Flower Auction Aalsmeer and Rail Terminal Hoofddorp (CTT, 1997)

3.2.2 Object oriented and process oriented decomposition

The OLS was a new and not yet existing system. Therefore, knowledge about existing AGV-systems and early designs for the OLS were used as system representations to enable decomposition into object-oriented and process-oriented system elements. In addition, expert sessions with designers of the AGVs and the logistic terminals were carried out to obtain additional information.

These different studies and information sources resulted in the following system elements using object oriented decomposition:

- AGV: vehicle moving around to pick up and drop of loads;
- Load: unit of materials to be moved by an AGV;
- Track: imaginary line between two points followed by an AGV to get from one place to another;
- Dock: machine that enables the movement of a load to or from an AGV;
- Dock place: physical configuration of a dock with one or more tracks for AGVs;
- Parking spot: physical configuration of one or more tracks that provides spots where AGVs wait for a new task, for example to pick up a load;
- Terminal: area consisting of one or more dock places and parking spots, which are connected by tracks to enable parking, loading and unloading of AGVs.

The process oriented decomposition resulted in 6 main processes, represented in Figure 3.3 to Figure 3.8. Figure 3.3 shows the process that is carried out for a load. Figure 3.4 shows the main tasks that are performed by an AGV. Figure 3.5 and Figure 3.6 are functionalities for allocating the scarce resources, e.g., AGVs and docks, to the entities that require these resources, e.g., loads. Figure 3.7 and Figure 3.8 show in more detail how the AGVs move safely over the available tracks.

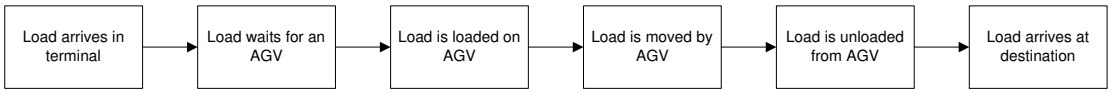


Figure 3.3: Process of a load

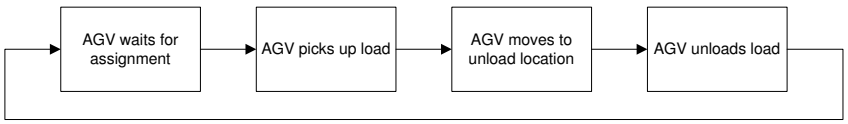


Figure 3.4: Process of an AGV

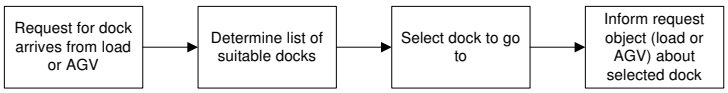


Figure 3.5: Process of allocating a load or AGV to a dock

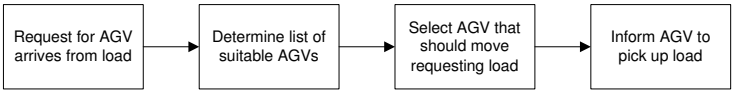


Figure 3.6: Process of allocating an AGV to a load

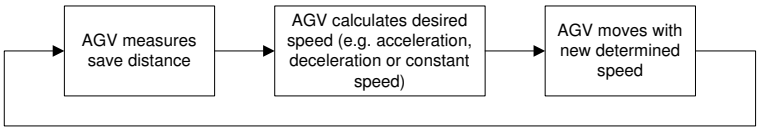


Figure 3.7: Process of an AGV driving

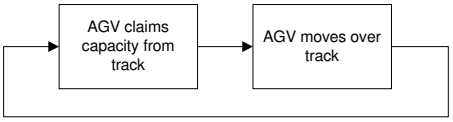


Figure 3.8: Process of an AGV moving over tracks

The safe distance between AGVs, as mentioned in Figure 3.7, was measured in the final system using sensors and a collision avoidance mechanism. The collision avoidance mechanism consisted of semaphores that restrict the use of tracks by AGVs (Lindeijer, 2003). Figure 3.8 shows how an AGV claims one or more semaphores before starting to move over a track. The system of claiming semaphores to gain permission to move is defined by the TRACES-concept (Transport Control Engineering System) for concurrent use of infrastructure (Evers and Koppes, 1996).

3.2.3 Existing domain specific extensions

Several generic simulation environments provide extensions for the domain of transportation systems with AGVs. We evaluated the suitability of domain specific extensions of the generic simulation environments Arena, eM-Plant and Automod. The key issue in this evaluation was whether the TRACES-concept (Evers and Koppes, 1996) could be implemented in addition to the track layout.

The domain specific extensions of the simulation environment Arena and Automod take care of safety mechanism internally. It is possible to interact with the safety mechanisms, but the TRACES concept can not easily be developed in a reusable model construct. The only way to overcome this is to use lower level constructs in these simulation environments, and not use the AGV systems that are provided. The generic simulation environment eM-Plant

allows model developers to add TRACES logic, but only if it is directly linked to infrastructure, so every infrastructure has one semaphore. This way of handling the TRACES semaphores was not acceptable, given that several tracks might overlap and use the same semaphore for safety. In addition, the domain specific extension of eM-Plant did not allow vehicles to accelerate and decelerate, but assumed that vehicles had a constant speed. For this project it was concluded that it would be more difficult to adjust the existing model constructs of the domain specific extensions than to develop a new set of domain specific model constructs.

3.2.4 New domain specific extension

Based on the identified system elements a new domain specific extension was developed for the OLS project. The number of model constructs that were part of this domain specific extension increased during the process of implementation and carrying out simulation experiments. Figure 3.9 and Figure 3.11 give an overview of the model constructs that represent system elements obtained from an object oriented and a process oriented decomposition.

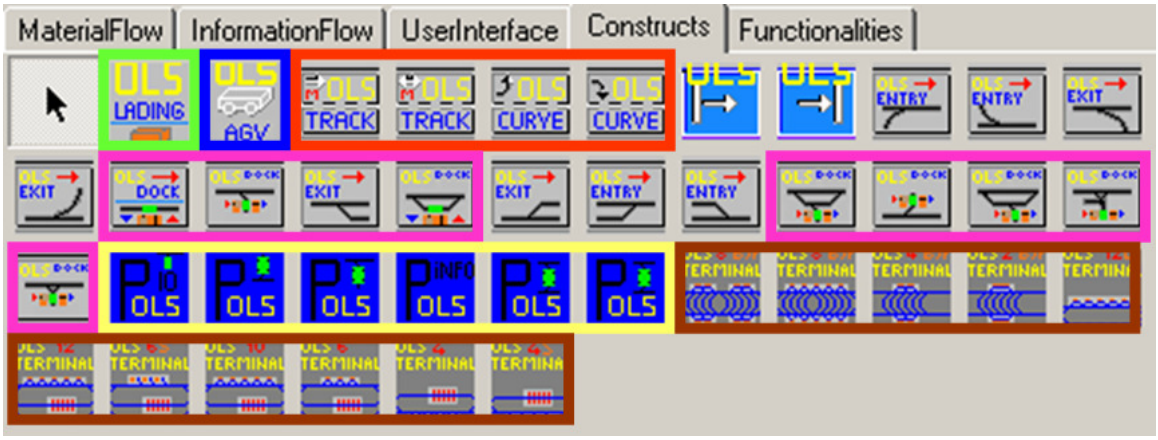


Figure 3.9: Model constructs that represent decomposed constructs

The thick lines in Figure 3.9 are added to show the relation with the objects identified for this domain. The first row of model constructs contains the load (first icon), the AGV (second icon) and the track (next four icons). The track model construct is further used in all model constructs to create parts of the layout of a terminal. The model constructs without any thick lines are compositions of tracks, for example a merger of two tracks into one track. The second row contains docks in different versions. The dock variants are composition of the basic dock configuration including one or more tracks to represent places where AGVs can temporarily wait for the dock to become available, see an example of the details in Figure 3.13. The last two rows contain more composed parts of a terminal, i.e. the parking (large P in icon)

and complete terminal layouts (remaining icons). The parking objects are composed of tracks with additional logic for parking. Some of the terminals of the bottom two rows are shown in more detail in Figure 3.14.

The system elements in the design of a terminal were not individual tracks, but layouts of a part of the terminal that were composed of several tracks, for example the configuration of the dock including the curves leaving the main track. This piece of layout was on its turn composed of several domain specific model constructs and made available as one new model construct. Figure 3.10 shows the composition of a specific dock with tracks, using several domain specific model constructs. Similar compositions have been made for other docks, parking spots, parts of terminals, and even whole terminals. As a result the set of model constructs of the domain specific extension grew quite large, as is shown in Figure 3.9.

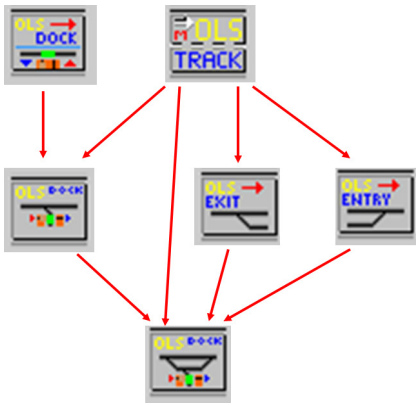


Figure 3.10: Composition of model construct "Side dock next to main track"

The AGVs and loads that use the infrastructure were controlled and managed by other model constructs. During the process oriented decomposition (Figure 3.3 - Figure 3.8) the allocation and safety mechanisms used in the AGV systems were identified, and these functionalities were implemented in different model constructs for the levels of control. Figure 3.11 shows the model constructs developed to represent the processes. The interaction between different model constructs represents the process as identified at page 43. For example, the model constructs in Figure 3.11 at the first row with a box around them are used for the process of an AGV moving safely over tracks (process in Figure 3.8).

Figure 3.11: Model constructs that represent decomposed functionalities

The processes were implemented in several ways. The allocation mechanisms and decision processes (see Figure 3.5 and Figure 3.6) were implemented as model constructs that make a match between the available resources and the requests. These model constructs are called “managers” and handle the allocation in the parking or the docks. For example, the terminal manager decides whether an empty AGV should be loaded with a waiting cargo, should stay at a parking spot, or should leave the terminal. Depending on the layout and size of the terminal, these managers make different types of decisions and use alternative information in their decision logic.

The AGV driving process, e.g., to determine their speed given their allowed distance, was divided into two model constructs. One model construct had the functionalities for calculating the distance, based on modeled sensor readings and permissions given by the TRACES safety concept, and one model construct had the functionalities to determine the new speed. Both of these model constructs were integrated into the model construct AGV and were dedicated to representing the required AGV behavior within this project. In the course of the project the way of handling these processes was changed, requiring coding adjustments in the process model constructs.

The last process, the process where AGVs gather access to new tracks based on their route (Figure 3.8), was modeled using a scripting language to define the steps that AGVs carry out. The scripting language enables flexibility when defining routes in the system, while still taking into account the required safety. An example of such a script that is part of the crossing shown in Figure 3.12 is given in Table 3.1. This script enables an AGV to move safely from left to right over the crossing. More about the implementation of TRACES in this domain specific extension can be found in Verbraeck et al (1998a) and Van der Heijden et al (2002).

Script LR	Comments
Insist SX	Claim ticket for crossing
Exec AX	Drive from left to centre
Exec XB	Drive from centre to right
Free SX	Free ticket SX

Table 3.1: Script for an AGV to move over the crossing in Figure 3.12 from left to right.

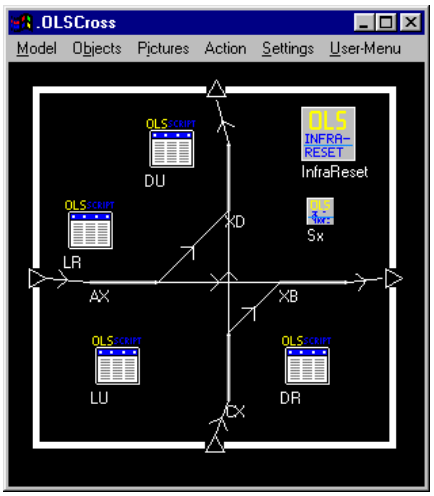


Figure 3.12: Crossing with scripts

3.2.5 Simulation study design of terminal layouts considering vehicle movement

Problem

Two simulation studies regarding the OLS system were carried out simultaneously. The first simulation study considered the complete system and applied optimization for the number of vehicles, the number of terminals and the effects of alternative forecasts for load patterns (Ebben, 2001). The second simulation study focused on the detailed behavior of AGVs, collision avoidance and routing, mainly within one terminal for loading and unloading of vehicles. Only the simulation study concerning the detailed behavior of AGVs within terminals used the domain specific extension discussed in this chapter.

Terminal layouts were designed according to several concepts for loading and unloading, parking and routing. Vehicle designs under consideration used various ways for loading and unloading the cargo. The different vehicle designs required specific concepts for docking to enable efficient handling of cargo and efficient use of the vehicles (Pielage, 2005). The variant layouts for docks are shown in Figure 3.13. The differences between the layouts were, e.g. the position of the vehicle, the direction for leaving the dock and the ability to park vehicles while another vehicle is being loaded or unloaded. The concepts for docking resulted in various possible layouts for docks in the terminal, alternative positions for vehicles to stop and routes towards and from docks.

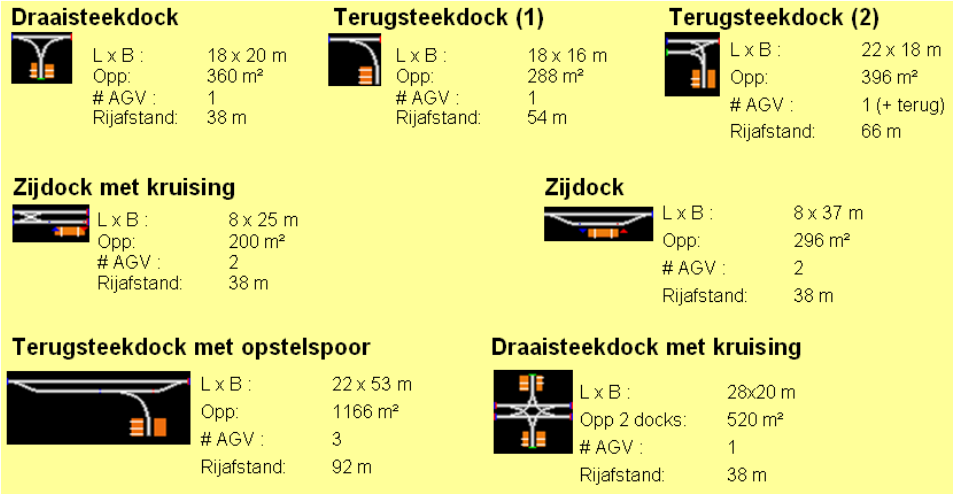


Figure 3.13: Dock variants with their Dutch names (Verbraeck et al, 1998b, p14)

The terminal designs differed in the docking concept used and in the way vehicles were parked and routed. Four of the many concepts of terminal designs that were available and that were evaluated using one or more simulation models are shown in Figure 3.14.

The evaluation of the terminal concept was the key issue in this simulation study. Depending on the concept an alternative safety mechanism, a number of docks and parking spots and the overall number of vehicles allowed in the terminal were evaluated.

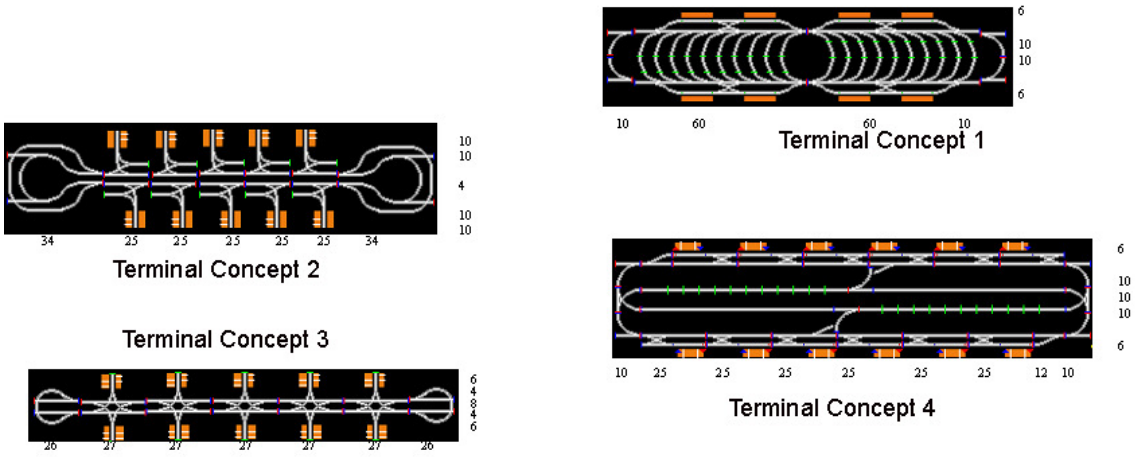


Figure 3.14: Different possible layouts of terminals that have been evaluated

Project approach

A traditional simulation study starts with evaluation of the original system and then defines alternatives (see figure 1.8). Because this study was about a new to be built system, it started with available alternatives that had to be tested. The solution systems were configurations of the terminal for many concepts of which four are shown in Figure 3.14. The evaluation of the feasibility of a terminal design was carried out in two steps. The first step was an experiment using a scenario of one peak hour in which a high number of AGVs arrive at the terminal to load and/or unload.

Successful terminal designs were further optimized with scenarios of the expected load pattern for a full day pattern of arriving and departing vehicles according to the 30th busiest day in the year 2015. This full day pattern included an early morning peak or a late afternoon peak. A good terminal layout should succeed in handling all loading and unloading AGVs in a reasonable time, partly during off-peak hours. The simulation experiments with the layouts provided insight into different distances for vehicles to travel, the ability to use parking spots, the number of crossing vehicles at a terminal and the utilization of docking places.

Simulation models of terminal layouts considering vehicle movement

Model constructs of the new domain specific extension were used to implement simulation models of terminals. These simulation models were

used to perform experiments to evaluate possible terminal designs. Figure 3.14 provides an overview of four successful terminal layouts that were used in experiments with full day scenarios. The terminal layouts have different shapes, different numbers and types of docks, different numbers and types of parking places, and a different control logic. The initial simulation models were easily instantiated once the model constructs with the layouts of the terminals were available, because the rest of the simulation model contained only a generator of vehicles, a track where vehicles were waiting to enter the terminal in addition to the model construct representing the terminal with its layout and control.

Verification and validation

The simulation models were verified by making a detailed evaluation of individual vehicles in the terminal. All events of an individual vehicle were observed to check that the vehicle was accelerating and decelerating at the right moments, claimed the right set of TRACES-semaphores and received the correct allocations from the terminal manager.

For verification, we mainly based ourselves on the animation of the simulation model. Each infrastructure model construct had an animation that showed exactly where the vehicles were. The model construct vehicle changed color depending on its state and it was easy to verify whether the vehicles received the correct state, i.e. accelerate or decelerate at the right moments.

Validation using data from a real system was not possible, because there was no real system. We performed validation sessions with experts of control systems and experts on the design of terminals to evaluate the correctness of the simulation models and the model constructs.

Experiments

With every terminal layout two experiments were carried out, one, a peak hour check and two, a full day experiment. The analysis of the output results of the experiments triggered several types of adjustments to the terminal design to try to improve the performance and check the sensitivity of the design to changes. A couple of the adjustments to the terminal designs dealt with:

- The number of vehicles allowed at the same time within the terminal. If too many vehicles were allowed in the terminal, congestion took place, the average speed of the vehicles dropped due to the applied safety mechanisms.
- “Smarter” TRACES scripts. Most terminal concepts (especially 1 and 4) were extended with additional TRACES scripts to improve the

throughput and enable vehicles to keep a higher average speed in the terminal.

- Process duration for loading or unloading AGV at dock. Various process durations were evaluated to see the effects of the docking time to the overall throughput.
- Required time for communication between AGVs and control systems. Anticipating on a slow communication process between the AGVs and the control system resulted in a more reliable system when actual communication times varied stochastically.

Initially the simulation models of alternative terminal layouts and their configurations provided just output of the number of AGVs that were loaded and unloaded and the average time AGVs stayed in the terminal. The output of these initial simulation models triggered requests for insight in many other system performance indicators. These system performance indicators were included in the simulation model by instantiating new model constructs and by adjusting existing model constructs. Examples of performance indicators that were added after the initial simulation experiments were: use of batteries, number of accelerations and decelerations in the terminal, time an AGV was waiting for a dock and utilizations of docks, dock places and parking spots.

Results of the simulation study terminal layouts considering vehicle movement

The experiments that were performed using the simulation models were based on a wide range of assumptions for vehicles, docks and load patterns. Most design options within the OLS project were completely open at the time that the simulation models were developed, because the design teams were just beginning to evaluate the possibilities. The simulation models that were used answered the question which terminal variant performed best, but the output of the evaluations of designs did not lead to a choice for a terminal due to the many assumptions that had been made for the model constructs of the domain specific extension. Therefore, the result of the simulation study was not a simple suggestion to use one terminal design, instead it was a set of design guidelines that should be considered in the final design of the terminals. This advice was supported by tables and graphs gathered from the performance indicators of all the experiments. Analyses of the experiments resulted in more than 30 guidelines for designers of terminals in the OLS system. The most important guidelines for the design of a terminal are listed below; additional guidelines can be found in Verbraeck et al (1998b)

- Avoid the AGVs crossing the main traffic route while entering or leaving a dock;
- Enable AGVs to be loaded and unloaded at the same dock to improve AGV throughput in the terminal;

- Provide parking spots very close to a dock to improve the throughput of the terminal and utilization of the docks;
- Enable AGVs to make turns in the terminal in as many places as possible to achieve short terminal times in wide terminals.

The main advice for the terminal layout was to design a terminal that avoids crossing traffic, that spreads the traffic over the space of the terminal as much as possible, and that provides a “slow” and a “fast” track, where the slow track is a side track used by vehicles that wait for their docking operation.

The following graphs, Figure 3.15, Figure 3.16 and Figure 3.17, are some examples of the data that was generated and automatically combined into an Excel sheet to visualize the system performance. Figure 3.15 shows the distances AGVs drive against the time they stay in the terminal. Figure 3.16 shows the average number of accelerations and decelerations by the AGVs for each hour of the day (important for battery usage). Figure 3.17 shows the waiting time for loads before leaving the terminal during a day, clearly showing two peaks times at the terminal.

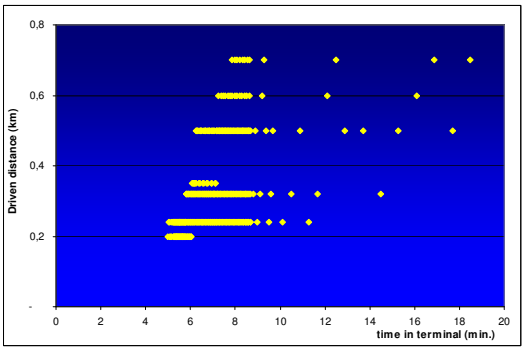


Figure 3.15: Relation between the terminal time and the driven distance of each AGV

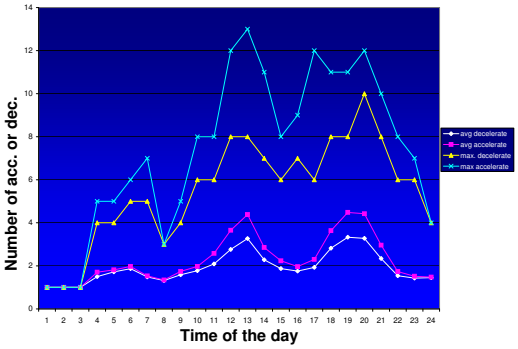


Figure 3.16: Number of times AGVs accelerate or decelerate during a day

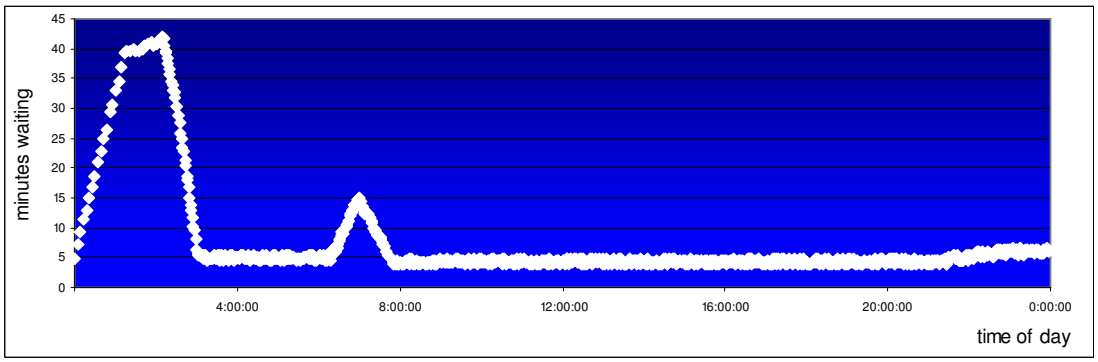


Figure 3.17: Time a load waits before leaving the terminal during a day

Observations simulation study terminal layouts considering vehicle movement

This simulation project provided insight into the problem owners in the design of vehicles, docks, terminals and allocation mechanisms. De Vos Burchart, one of the problem owners, stated in Van der Heijden et al (2002, p.19): "*One of overall goals in this project was to show the feasibility of an AGV based solution…. The contribution of the simulation group was essential to analyze the logistical performance of the system…. Simulation was extremely helpful in developing and testing our new control system.*" The satisfaction of problem owners is achieved thanks to the wide range of simulation experiments that could be carried out to evaluate terminal designs, control mechanism and allocation mechanisms.

De Vos Burchart confirmed that the simulation study was extremely helpful; however, several of the questions of problem owners, which were posed after the first publication of results (Verbraeck et al, 1998b), have not been answered. The most important questions that arose, after the report was published were the following:

- What would be the effect of an advanced or centralized allocation mechanism that links AGVs to certain docks, parking places or loads?
- Can alternative TRACES safety scripts improve the performance of terminal layouts that were not efficient?
- How will the terminal perform when it is integrated in the complete OLS system?
- What effects will adjustments of size, radius or acceleration of the AGV have on terminal layouts and performance?
- What relations exist between communication time, docking time, capacity of the terminal and number of parking spots or docking places?

These additional experiments could not be carried out, due to the time required for each experiment. Even though the model constructs of the domain specific extension would enable some experiments to be carried out fairly easy, other experiments would require more effort and some could not be implemented due to early design decisions taken during the development of the domain specific extension.

The simulation experiments have been used for evaluating and improving the control system of the individual vehicles and for empty vehicle management. In 2002 the research for the OLS ended, because the financial risks for the investments in building the OLS-Schiphol were considered to be too high (Van der Meer, 2002).

3.2.6 Observations on domain specific extension in case OLS terminal design

We made the following observations by linking this simulation study to the activities of Figure 1.8 and the expected benefits and potential risks of using domain specific model constructs listed in chapter 2:

Activity 1. Problem description & define conceptual model: the expected benefits and potential risks for conceptualization mentioned in chapter 2 were not observed in the use of the domain specific extension for AGVs, because the domain specific extension was developed specifically for this simulation study and problem description.

Activity 2. Select model constructs: each of the docks, parking spots and terminal designs were implemented as a model construct of the domain specific extension. The benefit of easier selection of model constructs was encountered in the simulation studies, because the translation of system elements to low level model constructs was not necessary. When someone designed a new piece of infrastructure the model construct to represent this system element could be composed from already available model constructs such as the track, dock and parking spot.

Unfortunately, for the control model constructs, composing a model construct from lower level domain specific model constructs was not possible. For example, the terminal manager was redesigned several times to make it suitable for the terminal layouts and configurations. The risk identified by Davis et al (2000) that system elements can not always be represented by existing model constructs in a domain specific extension was encountered during this activity. In chapter 2 we added this potential risk to activity 8 “Instantiate simulation model for identified solution”, but this case study showed that the risk was also encountered if a domain specific extension is developed specifically for a simulation study.

Activity 3. Data collection: the benefit of a known format did not apply to this case study, because the format required by the domain specific model constructs was adjusted and changed as the implementation of the model constructs progressed, just as it occurs in a normal simulation study where the need for data is not known beforehand. As a result the data gathering for this simulation study could not start directly at the start of the project, but was postponed until sufficient insight was gathered into the type of data that was required.

Activity 4. Instantiate simulation model for original system: the simulation models clearly contained fewer model constructs and were quite easy to develop initially. The composition of the layouts of the terminals was easy and fast, the TRACES-semaphores could easily be added and the initial versions of the TRACES-scripts for the vehicles were quickly defined. Once more difficulties arose regarding the model constructs that represented automatic management systems, but that was more due to the lack of availability of certain domain specific model constructs as mentioned for activity 2, than to the development of the simulation model.

The initial set of simulation models was instantiated by the developers of the domain specific extension. These developers understood the capabilities of the domain specific model constructs fully, and knew what to do when errors were observed during the test runs. Some additional simulation experts joined the project team later to instantiate simulation models of alternative layouts of terminals and docks. These model developers had not participated in the development of the domain specific model constructs and they had quite some difficulties when it came to understanding the full concept and the way it was represented by the model constructs. This was partially caused by the lack of example models and the fact that only technical detail documentation was available. The result was that the new model developers had difficulties with parameterization of the model constructs, which was also caused by a lack of simple interfaces and contextual help for the model constructs.

Activity 5. Verify and validate simulation model for original system: the initial simulation models were tested by the developers of the domain specific extension to test the model constructs. No formal testing was done before the simulation models were instantiated. Thus the benefit of reduced verification requirements because the model construct had already been tested did not apply. In the validation of the simulation models the visualization of the system, automatically provided by the model constructs, proved its added value. The visualization of the vehicles could be observed and analyzed at levels of composition, enabling us to view the visualization of the complete model, and simultaneously a visualization of for example an individual dock with the movement of vehicles around it.

Unfortunately, the benefit that calculations to support the statistics that problem owners are interested in are already available inside the model constructs did not apply in our case. The verification and validation sessions triggered what type of statistics were required and which ones should be added. Luckily these statistics could be added to the model constructs quite easily and were automatically shown the next time the simulation model was run.

The potential risk mentioned in literature of time consuming model adjustments after verification, has partially been encountered. There were quite some adjustments to be made, but because the model developers also developed the domain specific model constructs, they knew exactly how to tackle the differences that were observed during verification and validation sessions.

Activity 6. Analyze output of simulation model: we extensively used the benefit of standardizing the output of model constructs. Each model construct calculated some statistics and these were all combined and analyzed using one dedicated Excel sheet. This ability saved a lot of time for collecting and visualizing the statistics originating from several simulation models and thus clearly was a benefit.

The potential risk of lack of performance indicators was mitigated by adding extra performance indicators. Step wise the number of performance indicators was extended in the model constructs and thereby automatically included in the simulation runs.

Activity 7. Define solution for analyzed output: the model constructs had few user interfaces that were exposed to the modeler. The generic simulation environment does not allow for easy user interface development and thus this was left out of the model constructs. The benefit of receiving triggers for solutions by observing parameters was thus not observed. In addition, the problem owners and system experts who generated the majority of the alternatives had never seen the parameters of the model constructs. Therefore this benefit was not observed in this case study.

Activity 8. Instantiate simulation model for identified solution: a large set of experiments was carried out, because the model constructs of the domain specific extension enabled the model developers to:

- easily represent the infrastructure for the terminals, based on available model constructs, like the many alternative dock places and parking spots;
- easily compare the output of the simulation models, because each model construct provides the same type of statistical information, which was prepared as a standardized report;
- easily visualize the model, thanks to the detailed visualization included in all model constructs;
- easily apply different control mechanisms to safeguard terminal designs using instances of script model construct to represent the TRACES safety mechanism.

The benefit of parameterization did not apply during some of the experiments. The list of unanswered questions that was mentioned before, was partly caused by insufficient parameterization of the model constructs. For example, changing the diameter of a curve has effects on the safety controls and on the overall layout of a terminal. This change therefore still required several changes within the model construct. This difficulty would certainly have been observed when the simulation model would have been composed of generic model constructs, but this does not justify that the use of domain specific extension did not result in this expected benefit.

Changing the simulation model for some of the experiments also resulted in errors during the run of the experiment. Model constructs of the domain specific extension depended on each other in unexpected ways even though an object-oriented approach was used in decomposing the system.

Activity 9. Verify and validate simulation model for identified solution: the same expected benefits and potential risks applied as for activity 5: verification and validation of the simulation model for the original system.

Activity 10. Analyze output of simulation model for identified solution: the same expected benefits and potential risks applied as for activity 6: analyze output of the simulation model for the original system.

3.3 Exploratory case study 2: passengers at airports

3.3.1 Introduction: domain specific extension for passengers at airports

Simulation models are used by Amsterdam Airport Schiphol to support decision making processes with respect to extending and redesigning its passenger terminal. This case study project used a domain specific extension with model constructs for passengers at airports to avoid the need for simulation models to be developed from scratch in subsequent studies. This domain specific extension was developed based on experiences gained in an earlier project at the airport and described by Babeliowsky (1997) and Gatersleben and Weij (1999).

This section contains a description of the domain specific extension for passengers at airports and the use of this extension in three different simulation studies. Two of these studies were carried out at Amsterdam Airport Schiphol regarding an increase in passenger numbers, and a new check-in procedures for the airline KLM. The third simulation study was done at JFK airport in New York for analyzing the capacity of new check-in facilities.

3.3.2 Object oriented and process oriented decomposition

Airports all over the world have similar infrastructure and passengers go through similar processes at these airports. People move from an origin (either the airplane or the entrance of the terminal) through hallways, concourses and lounges towards a destination (either the airplane or the exit of the terminal) and before they reach their destination they participate in several processes, e.g. checking in, passport check, shopping, visiting restaurants and lounges, collecting luggage, and boarding. The activities that people perform at airports are reasonably standard, it is easy to generalize the systems behavior at different airports to a set of basic processes. However, airports have different ways of handling these processes, thus one needs to be able to parameterize and extend the basic processes to cover many alternatives. For example, at Amsterdam Airport Schiphol, the check-in process for a flight with a European carrier within Europe has fewer security issues, and thus requires less time and resources, than a check-in process for a flight to the United States.

People at airports use the infrastructure available within an airport for their various processes and to travel from their origin to their destination in the terminal. Arends (1999) made an overview of the infrastructure system elements that he regarded to be important within a simulation study of passenger flows in airports. He also performed a generalization of the

decomposed infrastructure objects. All system elements he identified were location dependent, and described an area at the airport where a certain function was performed. Examples are a check-in counter, a hallway, a gate, a seat and a shop. These system elements have been generalized further to one top-level model construct which he called an area. An area is a location where one or more passengers stay for a certain time, and sometimes carry out a certain activity. After this time the passengers try to move to the next area. The time for staying in the area depends on the type of infrastructure, the activity, and passenger characteristics. For example, in a hallway the staying time depends on the distance to walk, the number of other passengers in the area and the direction in which they are walking, and on the used surface (cart or not; family) and walking speed of the passenger. For a seat area as part of a gate area, a passenger will stay in a seat until a signal sounds that the boarding of a flight starts. In each of these areas the capacity is limited and thus each area can become a bottleneck for the processes at the airport when the number of passengers rises.

In the same way Arends (1999) has applied object oriented decomposition to identify types of people present at an airport. He made a detailed list of types of people, which he generalized to passengers, meet-greeters and personnel. Further he observed that people who are alone behave differently than people who are in pairs or groups. For example, a group of five co-travelers wait for each other after the passport check, take the same route, go to the same restaurant and arrive at the boarding at the same moment. Therefore passengers, meet-greeters and personnel are further generalized to groups.

By applying process oriented decomposition Arends (1999) identified system elements regarding the movement of people and the allocation of areas to flights or airlines. The process flow of people moving in the airport is divided into a detailed flow from area to area and a higher level flow from origin via intermediary destinations to a final destination. Further process flows were defined for the movement within areas, regarding the adjustment of capacity, and for the determination of the time spent in the area by people. Finally, the process oriented decomposition shows how the available capacity within the airport, provided by the areas, is allocated to different airlines and specific flights. These allocation mechanisms are also seen as specific processes.

The process of people entering and leaving an area is shown in Figure 3.18. The process is the same for all areas and it is assumed that this process applies for airports all over the world, as this is a basic and very generic description. The process of a group can be further specified depending on the type of area. For example, a passenger in a check-in area will have a staying time that depends on the speed of the check-in process, while if this

passenger is in a shopping area the staying time is a combination of the attraction of the shop and the spare time of the passenger.

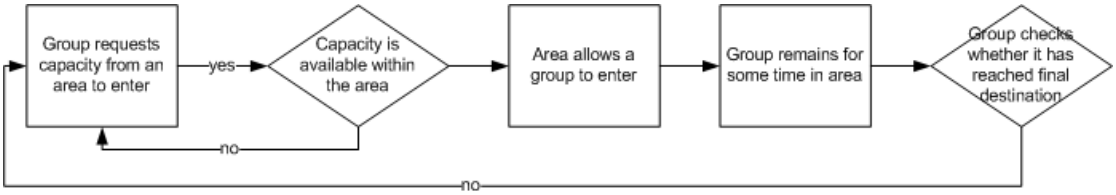


Figure 3.18: Process flow people

The destinations of a group, as referred to in Figure 3.18, depend on the purpose and moment of arrival of the group. Passengers that arrive by car and leave by airplane will perform process steps like check-in, go through passport check, security check and boarding an airplane. Some of these passengers might also have to check-in luggage, and some might have a Schengen[2] or national flight, which means that they do not have to go through customs. Passengers who arrive by airplane and transfer to another flight have a different process. Business class passengers behave differently from economy class passengers; individual passengers differ from families and larger groups. Therefore, the process for passengers at an airport can be different for each individual passenger at each airport. Decomposition of this process resulted in a set of process steps, of which some are shown in Table 3.2.

Table 3.2: Example process steps for routing behavior of passengers at airport

Process step	Description
Arrive	Entering airport
Board airplane	Leaving the airport via an airplane
Check of boarding card	Showing the boarding card to stewardess, often prior to enter restricted areas or airplane
Check of passport	Showing the passport to security employees, often prior to enter or leave restricted areas
Check-in for flight	Checking-in for a flight and optionally dropping off luggage
Enter gate	Moving to the gate where the passengers will depart

[2] The Schengen agreement is a treaty dating from 1985 between European countries which eliminates all internal border controls between them.

The passengers need to go to a certain area for each of the process steps. For example, the passengers need to select a check-in counter based on their airline or flight when they arrive, or the correct gate from where to board their airplane. The passengers receive information regarding their destinations from information providing objects such as monitors containing gate information.

3.3.3 Existing domain specific extensions

At the start of the project, two domain specific extensions were available for modeling passenger behavior at airports, 'IBM Journey Manager' (Bitault, 1997; Snowdon et al, 1998) and 'PaxSim' (Joustra and van Dijk, 2002). Both are extensions of the generic simulation environment Arena. These two domain specific extensions applied decomposition using the object oriented view. Some examples of model constructs available in those two domain specific extensions are a check-in counter, a gate and a security check. Between these main model constructs the passengers move freely and unrestricted. This does not fit with the concept of areas, because passengers can meet bottlenecks and queues anywhere at an airport.

The process for passengers is also too fixed. Passengers in these models go as quickly as possible to the gate, while normally passengers who arrive early at an airport spend some time hanging around and enjoying the shops or restaurants. No modeling concepts are offered in these domain specific extensions to represent flexible passenger processes, therefore these environments could not be used for the particular purposes of our simulation studies.

3.3.4 New domain specific extension

A new domain specific extension was developed for passengers at airports. This new domain specific extension includes model constructs to represent areas and a variety of processes for passengers. The model constructs of this domain specific extension were implemented in the generic simulation environment eM-Plant. The system elements that were identified during system decomposition by Arends (1999) were implemented using an inheritance structure. In this way the top-down implementation steps as suggested by Pater and Teunisse (1997) were performed.

The process steps regarding the detailed process of passengers in an area, see Figure 3.18, were implemented as functionalities of an area. The process steps regarding the directions of passengers at an airport, see Table 3.2, were implemented separately from the areas to keep the process of passengers flexible and easily adjustable. This separation of routing of passengers and use of areas by groups of passengers enables the use of areas by passengers with a different flow, for example departing and arriving passengers who use the same hallways.

The areas were developed into a structure with several variants of an Area that are inherited from the generic model construct called Area. The differences between the variants regard one or more of the following features and functionalities of the model construct: visualization, performance measurements, prioritizing of passengers, capacity and availability, determining the time a group will stay in the area and the restrictions for groups to enter. For example, a “check-in-counter area” calculates the time a group stays in the area for the check-in process using a statistical distribution that takes group properties as parameters, and in a “hallway area” the time a group stays is calculated using the walking speed of the group or passenger and the utilization of the area at the moment of entering. An overview of the most common Area model constructs is provided in Figure 3.19.

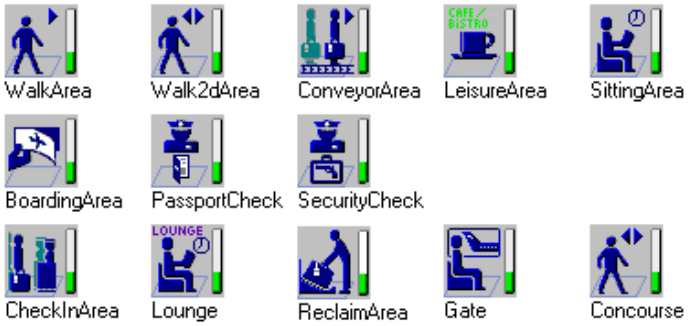


Figure 3.19: Model constructs to represent system elements of Areas

Each area in the inheritance structure provides a representation of one system element of an airport. Often standard combinations of two or more areas can be identified at airports. For example, the combination of a sitting area and a boarding check area can represent a system element “gate”. A system element gate is represented by a model construct called ‘ComposeArea’ where the required areas (in this example sitting area and boarding check) are combined. In the model construct ‘ComposeArea’ the individual behavior of the underlying areas is not changed, but additional statistics and hierarchical animation on the higher level are included. A GateArea can then be inherited from the ComposeArea. These model constructs increase the ease of model development by depicting repeated and recognizable parts of the infrastructure.

One of the assumptions in this domain specific extension is that each group can behave differently when going through the terminal at the airport. This assumption was implemented in the domain specific extension using functionalities for advanced passenger generation and scripting. Scripts describe the sequence and type of activities of passengers. The advanced passenger generation was implemented by a model construct that determines the number of passengers in the group, the speed of the group and many other passenger specific parameters. For example, a group that consists of

one individual experienced business traveler behaves differently than a group that consists of a family with three young children going on holiday. The routing behavior of these groups was represented in a script applying the decomposed functionalities mentioned in Table 3.2. Each script statement was a process that the group needs to perform somewhere at the airport and a combination of these script statements determines the routing of a group. Table 3.3 shows a simple example script of a group of passengers departing by an airplane. This script shows the sequence of processes for this departing group.

Table 3.3: Example of a simple script for a group departing by plane

Script statement:	**Argument(s):**
Check-in	Choose area based on flight number
Shopping	Choose area(s) based on interests and time left for shopping
Wait in gate for boarding	Choose area based on flight number
Enter airplane	

3.3.5 Cases where domain specific extension for airports is applied

The model constructs of the domain specific extension for passengers at airports were used in different simulation studies. Three different systems will be described that have been modeled using these model constructs. The three systems deal with different parts of airports and also with different levels of abstraction for the areas where passengers stay.

The first simulation study involved modeling the complete terminal at Amsterdam Airport Schiphol to evaluate passenger processes at a growth rate of 40% for the number passengers. Amsterdam Airport Schiphol aims to keep all their passengers within one terminal building and simulation models are used to evaluate different scenarios for extension (Arends, 1999). The second simulation study in which the model constructs of the domain specific extension were applied was the challenge from the KLM to divide their passengers better over their different check-in counters. Simulation should show the effect of different allocation mechanisms without reducing the passengers' freedom to check-in at any location of KLM designated check-in counters (De Witt-Hamer, 1999). The third simulation study handled the completely new check-in process at a renovated terminal building for international traffic at JFK in New York. The simulation study included determination of the desired number of check-in counters in this terminal and an evaluation of different ways to allocate check-in counters to airlines. (Heijman, 1999; Blom and Korf, 2000; Valentin et al, 2003a).

3.3.6 First simulation study: One Terminal Concept at Amsterdam Airport Schiphol

Problem

Amsterdam Airport Schiphol is an expanding airport that combines all its activities in one terminal building. The management of Amsterdam Airport Schiphol prefers to keep the one terminal concept when accommodating the increasing number of passengers. However, the growth in the number of passengers will have effects on passenger logistics, therefore airport management wanted insight into the allocation of processes to available infrastructure and the configuration of the terminal. Simulation models were constructed to show the effects of and Amsterdam Airport Schiphol's ability to handle the expected growth in the coming 20 years for the following design issues that the airport management was dealing with:

- physical separation of passengers of international and Schengen flights.
- determination of locations of shops and entertainment areas.
- new technologies for passport checks, such as biometric scans.
- using different floors for certain passengers in the terminal building.
- enlarge areas where passengers stay.
- personnel allocation for customs and passport check.
- personnel allocation for check-in and boarding.
- allocation of flights and airlines to check-in counters, gates and reclaim belts.

Simulation model One Terminal Concept at Amsterdam Airport Schiphol

Insight into the effects of all considered issues required a wide range of experiments with the simulation models. In this project, first a simulation model was built for the current system. This simulation model was validated with parameter settings taken from two different days. The validated simulation model was used to evaluate effects of alternatives and growth. The simulation model consisted of over 1500 instances of different domain specific model constructs representing areas and processes to accommodate passengers using this infrastructure.

The top overview of the whole airport terminal building is shown in Figure 3.20. The second more detailed level of one of the piers of the terminal is shown in Figure 3.21. The pier is composed of single areas, like WalkAreas and ConveyorAreas, and composed areas like a Gate-Area, which on its turn is a composed area representing a waiting area, a security check and the boarding process.

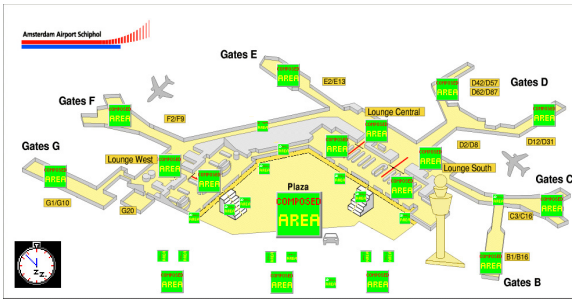


Figure 3.20: Model overview of Amsterdam Airport Schiphol

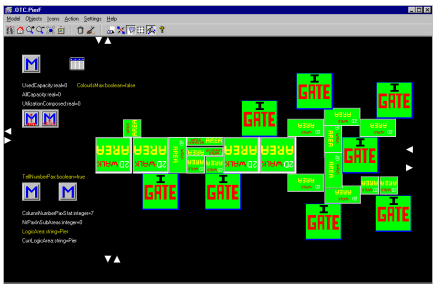


Figure 3.21: F-pier of Amsterdam Airport Schiphol

The modeled infrastructure was used by different types of groups that represent different types of passengers and visitors, as was explained before. The data used to define the unique specifications and behavior of groups was based on actual flight schedules and an analysis performed for the project in 1997 (Babeliowsky, 1999). The actual flight schedules that were applied to the simulation model were those for Friday 30th of April 2000, an average day with an average flight pattern, and Friday 2nd of July 2000, one of the busiest days in the year.

The insight for the management of Amsterdam Airport Schiphol was created by visualizations and statistical output generated by the simulation model. Visualization provided in the simulation model consisted of changing colors of the areas triggered by the utilization of the area. The more highly an area was utilized, the darker the color that represented the area, rating from light green to dark red. The statistical outputs were calculations of performance indicators included in each area in the simulation model. As the model contained more than 1500 areas, this resulted in a lot of detailed statistics. The main performance indicators that were used for analysis of the behavior at the terminal were:

- total time passengers waited, shopped or walked per flight;
- length of queues at the passport checks for departing passengers;
- length of queues for the central security boot at the different piers for departing passengers;
- number of passengers that were in one of the main halls at the same moment;
- percentage of passengers that missed their flight.

Verification and validation

The validation process of the simulation model consisted of verifying and validating the model constructs and the total simulation model. This study was the first study to be carried out using the model constructs of the domain specific extension for passengers at airports. Each new model construct was

tested for a short time in a small simulation model. The model construct was instantiated in the large simulation model of Amsterdam Airport Schiphol once the model construct had shown valid behavior in the small simulation model.

The second step for validation was to compare output of the model with measurements of the days under investigation. For validation, we counted the number of passengers and measured times for a large set of performance indicators during a full day at the airport. Afterwards we compared the measured data to the output of the simulation model for that day and we found several differences. After discussing the differences with experts at the airport, we concluded that the differences had to do with the following uncertainties regarding available input data:

- moment of arriving of passengers at the check-in process;
- the size of a group influences the handling times for the check-in process;
- the length of the queue influences handling times for the passport check;
- moment of arriving of passengers at the gate;
- activities of transfer passengers during their stay at the airport;
- airline companies use a specific capacity for the check-in process.

We concluded after consultation with the experts at Amsterdam Airport Schiphol that the simulation model was valid for experimentation.

Experiments

Evaluation of the output of the simulation models for future flight schedules showed that problems could be expected regarding the number of security checks that need to be carried out at different gates all over the airport and the queues that would result from an expected lower number of available security agents. One of the possible solutions that we evaluated using the developed simulation model was to combine the passport check and security for all departing passengers. The changes that needed to be made for experiments with this alternative control were an alternative model construct ‘gate’ and instantiate additional model constructs in the simulation model to represent the security check. These changes in infrastructure were easily performed, thanks to the structure of the simulation model and the existence of suitable model constructs.

The new simulation model with a security check directly after the passport check was evaluated with different number of passengers for the two evaluated days. Table 3.4 shows one of the main distinctive outputs of these experiments, the waiting time of passengers in front of the passport and security check. Figure 3.22 is a graph that zooms in at the value for security of

the south location and plots the average waiting time for groups of passengers (in seconds) against the hour of the day (0 – 24).

Table 3.4: Maximum waiting time for groups of passengers in minutes

Increase in number of passengers		0%	10%	20%	30%	40%	50%
Location	Process						
West	Passport	7.3	7.8	7.4	7.6	9.0	8.7
West	Security	0.6	0.9	0.9	0.9	1.0	1.0
Central	Passport	7.6	7.9	8.4	9.0	9.1	9.7
Central	Security	1.0	1.0	1.0	1.1	1.0	1.2
South	Security	3.5	3.8	3.8	4.0	4.2	4.2

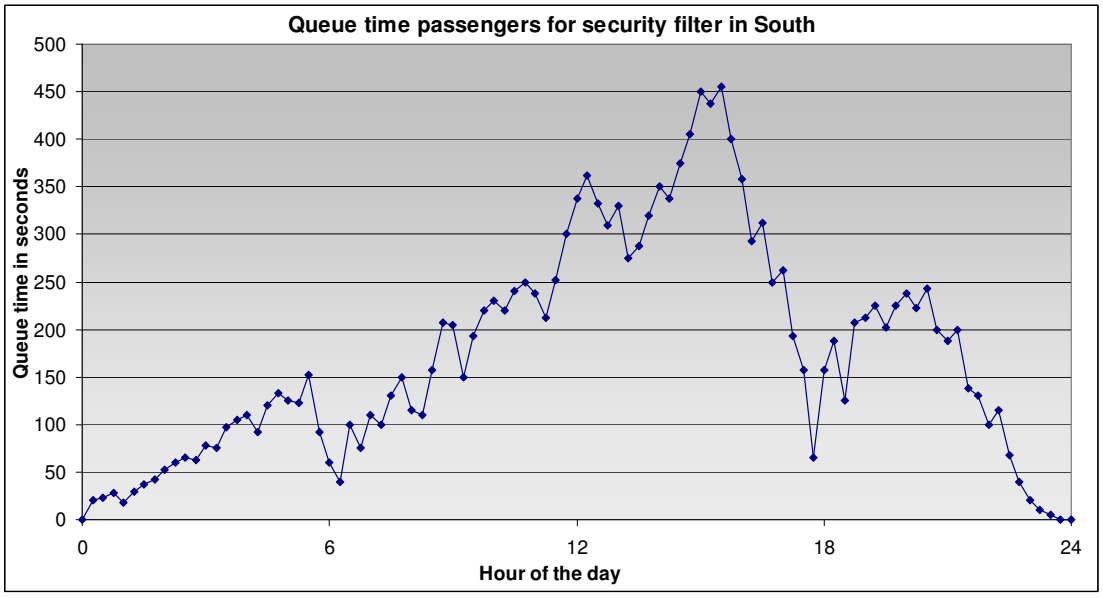


Figure 3.22: Waiting time of passenger at security filter in South

Results of the simulation study One Terminal Concept at Amsterdam Airport Schiphol

The output of the simulation experiments (Arends, 1999) showed that the waiting time of passengers at the different halls was within limits. The maximum queue time of 8 minutes was comparable with the current queue time. The merge of the passport and security check resulted in faster handling at the gate, therefore no extension of the number of gates was necessary and thus advanced infrastructure investments like more gates or double floor piers were not further evaluated.

Observations simulation study One Terminal Concept at Amsterdam Airport Schiphol

Thanks to the hierarchical concept and strong relation between system elements and model constructs, the first simulation model that was instantiated using the model constructs was developed in less than one day. This model included shortest path walking for passengers between destinations, scripts for passengers and allocation of flights to check-in counters and gates. We expected that the model development would go faster if model constructs were used, but less than one day was much faster than expected.

The first presentation of the simulation model to the problem owners triggered a lot of new questions, which could be answered using other model constructs to replace existing ones in the model. However, these model constructs were not available at the time, so in the next phase of the project the development of model constructs and the experimentation with the large simulation model went hand in hand. Time pressure resulted in many model constructs that were only tested and validated as part of the large simulation model. Every time that errors or problems were observed with the new model constructs this delayed the modeling process, and also frustrated the model developers who became more skeptical regarding the use of the domain specific extension.

Additional difficulties that the model developers faced were the growing set of model constructs. A tendency existed in the project to compose a new model construct for each alternative process. For example, only one type of check-in counter area was available in the domain specific extension that was used to compose the initial simulation model. Over time the domain specific extension was extended with versions that represented check-in counter areas for groups, versions using statistical distributions, versions using conditions for determining the handling time, versions including rules for changing capacity based on the number of passengers in queue, versions including rules for changing capacity based on the queue time, versions making distinctions for different types of luggage, and versions that combines several of the above behavior, e.g. statistical distributions and capacity changes. In addition the model developers included check-in counter model constructs that were pre-configured for the different terminals at Amsterdam Airport Schiphol, to reduce the parameter settings that needed to be done when a model construct was instantiated into the simulation model. Especially the combinations of features made the number of model constructs grow considerably, but also decreased the ease of maintenance as the same

behavior was implemented in different building blocks without a proper inheritance[3].

The model developers also came across a lack of data regarding future situations. The model constructs expected complete flight schedules as input data. The flight schedules should include the number of passengers, the location of the check-in counters used, and the gate for boarding each flight. This information was available for current situations, but not for (far) future scenarios. The model developers tried to make custom flight schedules, but that turned out to be more difficult than initially suspected, and led to double booked gates and check-in counters that were not used at all, where others had long waiting times. In reality, a careful planning process is carried out to make these allocations. The simulation model with the generated flight schedule showed passengers queuing very long and missing their flight, due to an impossible flight schedule. The model developers had a hard time creating valid simulation runs for future scenarios.

Finally, the model developers and problem owners had difficulties with understanding what was happening exactly in the simulation model. The visualization provided in the areas, i.e. changing colors, was not sufficient, while the performance indicators of each individual area were too detailed. The problem owners were interested in performance indicators like “how many passengers are too late for their flight” or “what is the minimum transfer time” but instead they received performance at a much lower level: “passengers have been waiting between 06:00 and 07:00 on average 0.03 seconds to enter WalkArea C24x5right1.” And this example output was duplicated over 1500*24 times as each area model construct in the simulation model provided this output for each hour of the day.

3.3.7 Second simulation study: KLM check-in allocation

Problem

KLM and partners use the check-in and baggage drop off counters in one of the three check-in halls at Amsterdam Airport Schiphol. Within this check-in hall KLM is free to allocate flights over the available check-in and baggage drop off counters. KLM allows every passenger to check-in at any available counter, but the monitors with check-in information show a dedicated location to spread the passengers equally over the different check-in counters to accommodate fast and easy handling of passengers. The allocation of flights over different check-in counters using the monitors has effects on the service levels for passengers of KLM flights. In 2000 the allocation of flights was

[3] The fact that certain aspects of different building blocks were changed in the same way to create children in the inheritance tree could only be solved with code duplication in eM-Plant. Multiple inheritance or aspect-oriented programming could have (partially) solved the issue, but is not available in this generic simulation environment. Furthermore, this would lead to a new inheritance tree of features, that was not foreseen in the abstraction and generalization of the system elements. It will be shown later how this issue was addressed in other studies.

performed based on the available time before a flight leaves. The question from KLM was whether alternative allocation mechanisms would improve the service level and reduce the costs of operations. This question was answered using discrete event simulation for the situation in 2000 and for future scenarios with more electronic check-in facilities.

Simulation model KLM check-in allocation

A simulation model was developed using the model constructs of the domain specific extension for passenger movements at airports. Specific model constructs were developed to match the system description of KLM. The set of model constructs of the airport domain specific extension was reduced to ease development, adjustment and use of the simulation model for this particular simulation study. The specification of model constructs for KLM concerned mainly visualization of the output of the check-in counters using graphs and animation of groups moving through the simulation model.

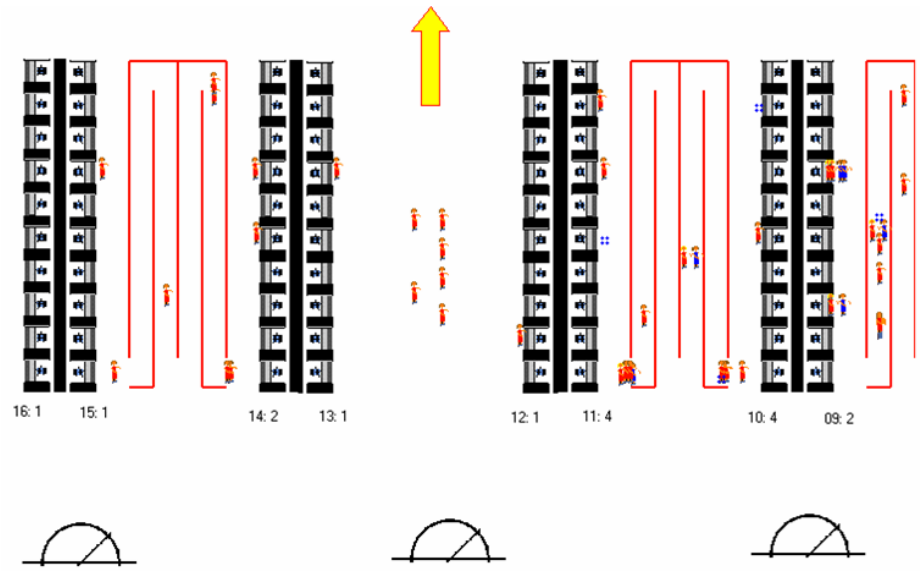


Figure 3.23: Simulation model of KLM check-in counters

The new model constructs were instantiated to develop a simulation model of the check-in area of KLM, see Figure 3.23. The bottom of the figure shows the entrance of the check-in hall and the arrow at the top indicates the walking direction for the passengers towards the passport check. The passengers enter the simulation model at the moment they enter the arrival hall and they leave the system when they start queuing for the passport check. The queuing process for the passport check, the ticket offices and passengers that walk through the check-in hall but do not use the facilities are not included in this simulation model.

KLM used a scheduling tool that determines the number of active operators in a check-in row, based on the flight schedule and additional

parameters like the expected percentage of passengers using electronic check-in and requiring just a baggage drop-off. The scheduling rules were translated to rules in the simulation model based on queue length. Every time the queue became longer than a certain number of passengers, an additional operator was requested to serve at one of the counters in a check-in row.

Verification and validation

The first processes of a simulation study, i.e. model development, verification and validation and analysis of the current system, were done as a group process. The group consisted of a developer of the domain specific extension who instantiated the simulation model, the user of the simulation model who would do all simulation experiments for evaluating the solution space, a domain expert that acted as problem owner on behalf of the KLM, and a facilitator.

The user of the simulation model and the problem owner made sure that during the model development process all desired input parameters for the model constructs were available. The problem owner used a face validation process to the instantiated simulation model during a group meeting. This face-validity was satisfactory and when the real data was compared with the output of the simulation model they judged that the simulation model was valid and ready to be used for experimenting to find alternatives in the solution space.

Experimentation

The simulation model was fed with historical data of passenger arrivals. The results showed that queues appeared at different locations during the day. Experiments were designed with different mechanisms to allocate personnel to check-in counters, and to inform passengers where they should check-in during the day. In the morning passengers arrived closer to the departure time of their flights, and thus a higher number of passengers use the last-moment check-in counters, while passengers for afternoon flights arrived well in advance. Different simulation experiments showed that the optimal service level would be reached by sending passengers that arrive between one and two hours in advance of their flights to go to check-in rows 10, 11 and 16 in the morning until 10 o'clock, and in the afternoon check-in rows 10 and 11 should be dedicated to check-in for flights up to 3 hours in advance. De Witt-Hamer (1999) describes many additional experiments that were carried out in 2000, for instance a number of scenarios with high e-ticket use by passengers.

Results of the simulation study KLM check-in allocation

The model building group sessions resulted in a valid simulation model that the KLM could use for additional experimentation. The experiments that KLM performed further (Wit-Hamer, 1999) provided new allocation mechanisms and resulted in a more sophisticated planning mechanism. The outputs of the simulation experiments and the new allocation mechanisms were adopted by the operational check-in department and have resulted in new allocations and scheduling of the check-in counters for KLM at Amsterdam Airport Schiphol.

Observations simulation study KLM check-in allocation

The possibility to compose KLM-specific model constructs made it very easy to carry out this simulation study. Instantiating the model constructs created a system representation directly, with representative animation and easy to understand performance indicators. In addition, experiments could be performed by straightforward parameterization of the input variables of the instantiated model constructs.

3.3.8 Third simulation study: Check-in counters at JFK Terminal 4

Problem

International terminal 4 at JFK airport in New York was completely reconstructed in the period of 1996 to early 2001, see Figure 3.24. The management of this terminal had opportunities to increase the number of flights from this terminal significantly by welcoming new airlines at the terminal after the construction period. The extension of the number of flights departing from the terminal would have effects on the expected service levels, mainly at the check-in processes at the terminal. A simulation study was carried out to evaluate the service level at the check-in area to consider possible future scenarios for growth in the number of departing flights.

Figure 3.24: Artist impression of the building of JFK International Terminal (Heijman, 1999)

Simulation model check-in counters at JFK Terminal 4

The simulation model was instantiated from model constructs of the domain specific extension for passengers at airports. The scope of the simulation model is marked by the box in the middle of Figure 3.25. The check-in hall at terminal 4 was represented by areas for the check-in process and walk areas. The walk areas allowed passenger to move from the entrance of the check-in hall to the check-in counter and out of the check-in hall towards the gates and the shops.

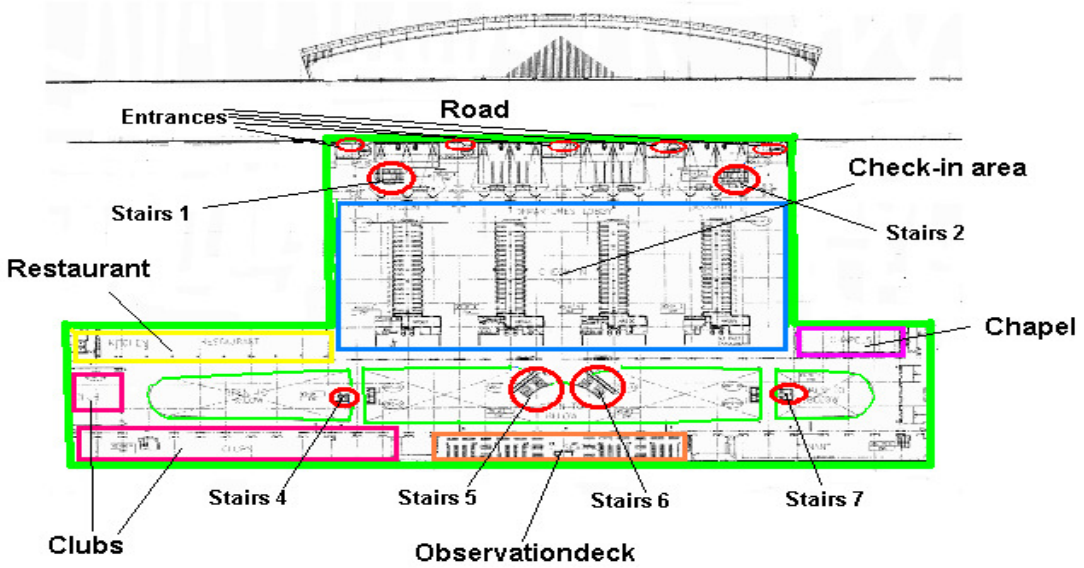


Figure 3.25: Map of level 2 of terminal 4 (Blom and Korf, 2000)

Check-in times differ considerably between different airlines. Handling time measurements were carried out for all different airlines. The processing times were lumped together on destination, based on these measurements, see Table 3.5.

Table 3.5: Processing times passengers check-in counters

Destination	Domestic USA	Europe	Far East	South America	Other
Processing time (min, mean, max)	2 min.; 2½ min.; 3 min.	2½ min.; 3 min.; 3½ min.	2½ min.; 3½ min.; 4 min.	3 min.; 3½ min.; 4½ min	3 min.; 4½ min.; 6 min.

The scripts to describe the processes of the passengers were kept simple and consisted of only four steps: “arrive at the airport”; “check-in”; “say goodbye”; “leave to the secured area”. The passengers could arrive at several entrances to the terminal. 70% of the passengers in the model arrived through the main entrances (see Figure 3.25), with 30 percent arriving by the JFK monorail system and then coming down the stairs. Business class passengers

were expected to arrive 30 minutes to 2 hours in advance of flight departure, economy passengers to arrive between 1 hour and 4.5 hours in advance. These passenger arrival times were based on measurements and questionnaires. Additional questionnaires among airlines that fly from this terminal showed that the patterns for economy and business class passengers apply for all airlines.

The flight schedules that were used in the base scenario were constructed based on projections for future growth and consisted only of predictions for airlines currently operating from the terminal. Four additional scenarios were created, each with an increasing number of new flights departing from terminal 4.

The domain specific extension was extended with a model construct for this system to allocate available check-in counters to departing flights, in line with the specific situation at this terminal. This allocation was also used to direct passengers to the correct check-in counter. This model construct took restrictions in the infrastructure into account and decided how many check-in counters would have to be available for the check-in process for a flight. The decisions of this model construct were based on the priority of airlines and the expected number of business and economy passengers to check-in.

Verification and validation

The verification and validation activities that were performed for this simulation model were restricted to a number of tests with extreme input parameters. Examples of the extreme input parameters that we used were a low number of check-in counters and a high number of passengers on a selected flight (Valentin, 2002).

Experiments

The first simulation runs showed that the planned check-in counters would not be sufficient for the expected increase in the number of flights. Changes in the allocation mechanism for allocating check-in counters to airlines resulted in an improvement of the waiting times, but still resulted in long queues. Alternatives that were evaluated included an increase in the number of check-in counters.

Results simulation check-in counters at JFK Terminal 4

Based on the output of this simulation study the management of Terminal 4 at JFK decided to increase the number of check-in counters and to do additional research into how they would allocate the available check-in counters to the airlines in the future.

Observations simulation study check-in counters at JFK Terminal 4

The simulation study provided the insights that the problem owners of the terminal were lacking. The model development using the model constructs also went well; however, there were some minor problems. This simulation study started with the domain specific extension that was tested and enriched during the two previous simulation studies mentioned. In these studies new model constructs were added to the simulation environment and this made the number of available model constructs larger. As a result, the model developers now had difficulty selecting the best model constructs to represent their system elements.

Another encountered risk was that the layout of the terminal showed 8 rows of check-in counters. Several airlines would share facilities in one check-in row. The processes at the individual counters could easily be represented by the available model constructs, but in this case no control was initially available to open or close additional check-in counters when needed, or to pick passengers from a shared queue. It actually required quite some time to develop the dedicated model constructs for these controls, and the use of the domain specific extension might have been equally or less effective than a simulation study using a generic simulation environment.

Finally, it turned out that using the domain specific extensions had a steep learning curve, and it took the simulation experts with quite some airport knowledge a lot of time to understand the way the model constructs should be used (Valentin, 2002).

3.3.9 Observations simulation studies domain passengers at airports using domain specific extension

Based on the activities of Figure 1.8 and the expected benefits and potential risks outlined in chapter 2, we learned the following from the three experiments using the domain specific extension for airports.

Activity 1. Problem description & define conceptual model: the problem descriptions in the three airport simulation studies were independent of the capabilities and available model constructs in the domain specific extension. In the first simulation study, a basic domain specific extension was available. In the other two simulation studies it was known beforehand that additional model constructs were required.

Activity 2. Select model constructs: for each of the three simulation studies, the model developers could select the model constructs based on the decomposition of the original system, because the model constructs and system elements were similar. The model developers did not need to make a translation from system element to a variety of model constructs.

Another benefit was that the different domain specific model constructs represented were part of an inheritance structure. The selection of the correct

model construct could be made based on functionalities that were related to the inheritance tree of the domain specific extension. The existence of this tree made it easier to select the correct model construct, although the set of model constructs was quite large in the end.

One of the newly observed risks was that model developers instantiated a new model construct as soon as they thought that a system element was slightly different from the offered model constructs in the domain specific extension.

Some of the system elements were represented by the wrong model constructs, due to the large set of model constructs available to represent check in areas and other areas that were available. The structure of the domain specific extension and inheritance helped in selecting, but did not prevent model developers from selecting the wrong model construct.

Activity 3. Data collection: the data collection was a large issue for the first simulation study, as there was no information regarding the flight schedule for 2020 on the level that we needed for the simulation study. The participants in the KLM check-in simulation study were informed during the first session, so in the second session they brought all the data that was required. In this simulation study there was a clearly defined benefit from using predefined data requests.

Activity 4. Instantiate simulation model for original system: the initial simulation model for all three simulation studies was developed rather easily and fast. The benefit of faster model development was clearly observed. The problem owners had insight directly and directly started to make suggestions on how to improve the simulation model. In a normal simulation study the problem owners would not have that much insight into the model and would not interact in the model development phase, but thanks to the model constructs the model was understandable and the problem owners recognized the layout from their maps.

The risk for correctly configuring the model constructs was encountered for the model constructs that generated new groups of passengers or triggered new destinations in the airports, such as the model constructs used to represent information screens for guiding the groups of passengers through the infrastructure. These model constructs required quite some data that was related to the flight plan, but not as obvious as a map of the airport or duration for a process. With some additional training and a demonstration in a small simulation model, the model developers sufficiently understood the importance of these model constructs and could configure the model constructs for the logical routing of passengers.

Activity 5. Verify and validate simulation model for original system: there was insufficient testing of the first version of the library of model constructs, and this resulted in several issues during verification process of the simulation study for Amsterdam Airport Schiphol. The fact that the model developers were testing the model constructs instead of their simulation model caused a

reduction of trust in the quality of the model constructs with the problem owners.

The model developers did not allow time for full testing and in two situations started to make their own adjustments to the model constructs at moments that errors were observed during the model development. Once the model constructs were completed the simulation model still behaved differently, because adjustments to the model constructs in the domain specific extension no longer had an effect on the model constructs instantiated in the simulation environment. This risk was mainly encountered due to the openness of the simulation environment used, allowing the model developer to change the logic of the domain specific model constructs.

Activity 6. Analyze output of simulation model: the model developers had the possibility to define a large amount of areas, a lot of group types, and different kinds of mechanisms to manage passengers and flights. Each of the model constructs that was used to represent the behavior of these system elements also collected a bunch of statistics. The model developers were overwhelmed by the amount of information that the system produced and it took them a lot of time to find out how to handle the data.

Activity 7. Define solution for analyzed output: the solutions that were defined were prepared, and confirmed by the analysis of the output data. The benefit of the parameters of the model constructs was not really confirmed, but the feared risk of limiting the model developers has not been encountered during the simulation study.

Activity 8. Instantiate simulation model for identified solution: the simulation models could be instantiated and parameterized for the new simulation experiments. Unfortunately, the main set of experiments of the first simulation study required a lot of additional data to be defined, mainly a valid plan for flights and use of gates. Such a plan was not available and thus was it very hard to perform the experiments for the future situation. This is not caused by a potential risk of use of domain specific extensions, it would also have occurred in a simulation study where only generic model constructs would have been used.

Activity 9. Verify and validate simulation model for identified solution: the same expected benefits and potential risks applied as for the verification and validation of the simulation model for the original system.

Activity 10. Analyze output of simulation model for identified solution: the same expected benefits and potential risks applied as for the analyze output of the simulation model for the original system.

3.4 Benefits and risks in the case studies

The simulation studies that were carried out in the two domains provided us with new insights into the effects of using a domain specific extension to compose simulation models. First of all we enjoyed many of the expected benefits that were predicted in literature (chapter 2). The simulation studies

were performed correctly and the problem owners were satisfied with the result.

The use of model constructs of the domain specific extension was not a complete victory. The concepts and approach helped with developing model constructs for a domain specific extension, but we encountered still most of the risks that we had identified from literature. We could mitigate most of the risks reasonable easily, because we were prepared for them, but the time and effort spent to mitigate the encountered risk in the simulation study could better have been used to improve the quality of the insight generated for the problem owner by the simulation study.

A summarizing overview of the benefits and risks that were introduced in chapter 2 and whether or not they were observed in the simulation studies in the domain is provided in Table 3.6. If “No” is filled in for a potential benefit it means that in the case studies for this domain we did not observe effects of the expected benefit. This is not negative, but points out that the case study has the potential of being even more effective. Potential risks that we did not observe during the execution of the case study (“No” in the table) probably did not occur and thus that the potential risk has been avoided by the way the domain specific extension was designed, structured and used.

In Table 3.6 we also describe the effect of the benefit or encountered risk to the simulation study. The effects of the benefit are described in relation to carrying out the same simulation study using model constructs of a generic simulation environment. The effects of encountering the risks are mainly the way that the risk has been mitigated.

Table 3.6: Summary of benefits and risks in the case studies

Expected benefits and potential risks	**Observed in case AGVs**	**Observed in case Airports**
Activity 1. Problem description & define conceptual model		
Benefit 1.1: conceptualize system elements with model constructs in mind - faster conceptualization - conceptualization is better prepared for model instantiation	Not applicable	Yes
Risk 1.1: scope of model developer is limited by model constructs	Not applicable	No
Activity 2. Select model constructs		
Benefit 2.1: no translation between system elements and model constructs - reduction of complexity	Yes	Yes

Activity 2. Select model constructs		
Risk 2.1: lack of trust results in no motivation to use domain specific extension - talk to model developers to explain potential benefits - show use of model constructs in other project	No	Yes
Risk 2.2: lack of insight in model constructs results in ignore domain specific extension	Not applicable	Not applicable
Risk 2.3: use of model constructs that are not suited for representation of system elements - training of model developers - include additional terminology in interface of model construct	No	Yes
Activity 3. Data collection		
Benefit 3.1: collection of predefined input data	No	No
Activity 4. Instantiate simulation model for original system		
Benefit 4.1: less model constructs used - simulation model is instantiated faster - simulation model seem better understandable	Yes	Yes
Risk 4.1: model developers do not understand model construct - training of model developers how to use model construct	Yes	Yes
Risk 4.2: model developers do not know how to parameterize model construct - training of model developers how to use model construct - additional terminology in interface of model construct	Yes	Yes
Risk 4.3: difficult to compose simulation model, because model constructs are not available - create additional model constructs to satisfy need in simulation study	Yes	Yes
Activity 5. Verify and validate simulation model for original system		
Benefit 5.1: no more detailed testing - reduction of time spend by model developer	No	No
Benefit 5.2: easily gathering validation data - reduction of time spend for data gathering	Not applicable	Yes
Benefit 5.3: structured and standardized performance indicators - easier gaining insight in performance of a system	No	Yes
Risk 5.1: mistakes of model developer are hard to overcome - training of model developers how model constructs were meant to be used - spend extra time to verify simulation model	Yes	Yes

(continued at next page)

Activity 5. Verify and validate simulation model for original system		
Risk 5.2: model developers know something is wrong, but cannot identify what to do about it - training of model developers how to interpret results - extra performance indicators and animation in model constructs	No	Yes
Activity 6. Analyze output of simulation model		
Benefit 6.1: structured and standardized performance indicators - easier gaining insight in performance of a system	Yes	Yes
Risk 6.1: model constructs do not provide performance indicators problem owner desired - extend model constructs with extra performance indicators	Yes	Yes
Activity 7. Define solution for analyzed output		
Benefit 7.1: model developers are triggered to find new solutions by parameters - more insight gained thanks to more experiments	No	Yes
Risk 7.1: model developers are triggered to find new solutions by parameters	Yes	Yes
Risk 7.2: model developers are limited by parameters and model constructs - extend model constructs with extra input parameters	Yes	Yes
Activity 8. Instantiate simulation model for identified solution		
Benefit 8.1: easy adjustment of model thanks to user interfaces of model constructs - easier to prepare simulation model for experimentation	Not applicable	Yes
Risk 8.1: solution is identified that can not be represented by model constructs - Ignore possible solution - extend model constructs and set of model constructs	Yes	Yes
Risk 8.2: adjustments of model constructs required to represent solution are time consuming - spend required time to fix model	Yes	Yes

The mentioned expected benefit and potential risks were not applicable for the AGV project for activity 1 “problem description & define conceptual model”, because the model constructs had not been developed when the simulation study started. In the simulation studies of the passengers at airport the model constructs were already available, but model developers demanded additional model constructs instead of limiting the conceptual models.

The risks resulting in ignoring the domain specific extension in activity 2 “select model constructs” were not encountered in these simulation studies,

because the model developers were forced in these simulation studies to use the domain specific extension. They could not ignore the domain specific model constructs and received additional explanations until they were convinced that the domain specific extension was the best way to carry out the simulation studies.

The OLS system and the future terminal at JFK did not exist at the time of the simulation studies. Therefore it was not possible to collect validation data and thus the benefit of easily gathering validation data did not apply to the domain specific extension for AGVs. In the simulation studies for the passengers at Amsterdam Airport Schiphol and the KLM the validation could be collected and in those studies the benefit regarding validation was observed.

The benefit of easy adjustments thanks to the interface of model constructs did not apply to the use of the model constructs of the domain specific extensions of the AGVs. The models could easily be adjusted, but the model constructs did not have a user interface. The changes were made directly in the logic or the attributes, not via a dedicated user interface. The model constructs of the domain specific extension for airports were extended with user interfaces. In the simulation studies using these model constructs the benefit was applicable and observed.

The risk of ‘Model developers are triggered to find new solutions by parameters’ is special, because to this encountered risk no counter actions have been carried out. The reason is that this risk is encountered after the experimentation and solution finding is finished.

3.4.1 New benefits and unexpected risks

We made more observations during the case studies that can be generalized to additional benefits of using domain specific extensions in a simulation study. These benefits had similar effects as the expected benefits identified in chapter 2. We also encountered some unexpected risks. These risks have been mitigated by extra activities. The new benefits, the unexpected risks and their respective effects and activities to mitigate them are listed in Table 3.7.

For the new benefit 8.2 and 8.4 in activity 8 “Instantiate simulation model for identified solution” we have to give a remark: “Partly” because these benefits applied in the majority of the changes that were made to the infrastructure of the AGV terminals or the areas in the airport. Unfortunately, these benefits did not apply to the model constructs for controls or processes. The model constructs as developed in the domain specific extension did not allow for composition of a control or process system and the model constructs could not be replaced easily, because other instantiated model constructs

were connected and depended on the originally instantiated model constructs. Thus, we observed benefits, but not as consistently as expected.

Table 3.7: New benefits and unexpected risks of use of domain specific extensions in simulation studies

New benefits and encountered risks	Observed in case AGVs	Observed in case Airports
Activity 1. Problem description & define conceptual model		
No additional benefits or risks have been observed for this activity		
Activity 2. Select model constructs		
New benefit 2.2: compose model constructs from developed domain specific model constructs to represent system elements - more flexibility during model development - easier to develop models	Yes	Yes
New benefit 2.3: easy selection of model construct thanks to structure of domain specific extension - faster model development	No	Yes
New risk 2.4: system elements can not be represented by model constructs - develop very specific model constructs	Yes	Yes
New risk 2.5: compose model constructs from developed domain specific model constructs only applied for infrastructure system elements - develop very specific model constructs	Yes	Yes
New risk 2.6: model developers can adjust internal logic of model constructs - Change internal logic back to original state	No	Yes
Activity 3. Data collection		
No additional benefits or risks have been observed for this activity		
Activity 4. Instantiate simulation model for original system		
New benefit 4.2: model development faster and easier	Yes	Yes
New benefit 4.3: model development by simulation novices - simulation model can be developed by persons who are no simulation experts	Yes	Yes
New risk 4.4: difficult to compose simulation model by person other than developer(s) domain specific extension - training by developer of domain specific extension	Yes	Yes

Activity 5. Verify and validate simulation model for original system		
New benefit 5.4: semi-automatic reporting of performance indicators - easier to gain insight in performance indicators - easier to report on performance indicators of simulation model	Yes	Yes
New benefit 5.5: observe animation at different levels of the composition: high level and at individual model construct - easier to gain insight in system - possible to gain top-down insight in system	Yes	Yes
Activity 6. Analyze output of simulation model		
New benefit 6.2: semi-automatic reporting of performance indicators - easier to gain insight in performance indicators - easier to report on performance indicators of simulation model	Yes	Yes
Activity 7. Define solution for analyzed output		
No additional benefits or risks have been observed for this activity		
Activity 8. Instantiate simulation model for identified solution		
New benefit 8.2: easy adjustment of model thanks to replacement of model constructs - faster preparation of simulation model for experimentation	Partly	Partly
New benefit 8.3: easy visualization thanks to incorporation of visualization in model constructs - faster model development	Yes	Yes
New benefit 8.4: composition of new model constructs enabled new solutions to be evaluated - flexibility in performing simulation experiments and thus more insight in system	Partly	Partly
New risk 8.3: replacement of model constructs causes errors in model constructs that were linked or connected. - spend required time to fix model	Yes	No

4 Testing domain specific extensions in a laboratory setting

4.1 Introduction

The use of model constructs of a domain specific extension in case studies have shown that participants encountered some of the known and new risks. The observations also resulted in conclusions regarding the advantages of using model constructs of a domain specific extension instead of using model constructs of a generic simulation environment. The case studies have been executed using the model constructs of new developed domain specific extensions. The case studies do not prove that they were more effective than if the simulation studies would have been carried out using model constructs of a generic simulation environment, for the simple reason that the size and availability of experts to carry out the simulation study did not allow to perform the case studies in two ways.

To date, very little research has been published that compares the use of domain specific extensions with the use of model constructs of generic simulation environments for problem solving. This type of research is provides additional insight to answer the question regarding why model developers do not use domain specific extensions. Three laboratory experiments, in which 80 novices and experts in simulation used simulation models to answer questions for a public transportation case, are described in this chapter. The simulation models were developed using either a domain specific extension or a generic simulation environment, so that a comparison could be made between the two types of model constructs.

The results of this comparison will be a first confirmation of the mentioned risks and provide the first insight into the causes and ways to mitigate these risks during a simulation study. The laboratory experiments have been simplified compared to a real-life simulation study, but because of these simplifications we can better identify the existence of risks described in literature and their causes. In this chapter we describe the main findings of the laboratory experiments. The full detailed analysis of the laboratory experiments has been described in a number of papers (Kolfschoten et al, 2006; Kolfschoten et al, 2010; Valentin et al, 2003a; Valentin et al, 2003b).

4.1.1 Model development for problem solving

The laboratory experiments that are described in this chapter investigate the difference between using generic and domain specific extensions on the effectiveness of a simulation study. In the laboratory experiments we measure how much of the required insight for the problem owner is provided by the

model developers. The effectiveness of the simulation studies carried out with domain specific or generic simulation environments is the gap between the requirements of the problem owners and the insight that the model developers provide during the laboratory experiments.

In the laboratory experiments the model developers had several ways to provide insight to problem owners. In the first laboratory experiment we only focused on the *number* of simulation experiments they could perform. In the second and third laboratory experiment the model developers had the freedom to provide insight in the way they considered best. In the second and third laboratory experiments the model developers had total modeling freedom for animation, level of detail, and types of performance indicators.

The objective measurement in the laboratory experiments was the "number of questions answered with required quality", see the block on the right of Figure 4.1. In the first laboratory experiment the answers were provided as quantitative output of simulation experiments, in the second and third laboratory experiment the model developers also used other ways to answer the questions. The simulation experiments in the second and third laboratory experiment dealt with parameter settings, policy change, and influences in values of data input and sensitivity analysis to values of performance indicators.

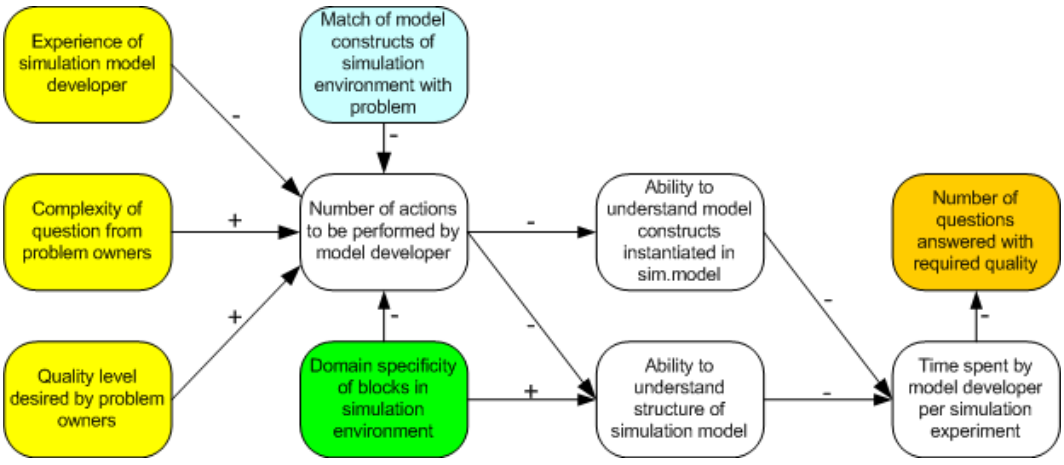


Figure 4.1: Causal diagram of problem solving with simulation models

The open blocks with their causal relations in Figure 4.1 are the intermediary steps the model developer had to take during the problem solving process. The diagram shows the relation between understanding a simulation model and the answers provided to the questions of the problem owners. Based on Pater and Teunisse (1997) and Diamond et al (2002) it is assumed that having fewer actions will improve the understandability of the simulation model. The improved understandability will reduce the amount of time that needs to be spent performing a simulation experiment.

The main variable that was varied in each of the laboratory experiments was the domain specificity of the simulation environment that the model developers were using. The system used in the laboratory experiments was public transportation using light rail. This system can be modeled using a generic simulation environment that takes into account resource allocation, or using a domain specific extension, designed for systems that fit the problem domain rail networks and passengers. The domain specific extension was specifically developed for the laboratory experiments. The questions of the problem owners within the laboratory experiments were taken into account when the model constructs for the domain specific extension were developed.

In the causal diagram (Figure 4.1) it is shown that the match between model constructs and the problem has an effect on the number of actions to be carried out. The model developers need to make adjustments to the model constructs if a system element can not be modeled with the provided implementation. This requires a lot of insight into the internal working of the model construct and quite some time to gather this insight and make the changes. Adjusting a model construct of an existing domain specific extension requires detailed insight into the way the model constructs of the simulation environment are implemented. It was in this case not possible to find volunteers with enough time and competences and skills in the field of discrete event simulation.

The three causal blocks at the left-hand side of Figure 4.1 are the variables that were varied between the three laboratory experiments. “Experience of simulation model developer” was evaluated using novices for the first and second laboratory experiment and experts for the third laboratory experiment. The participating novices were engineering students that had followed courses on discrete event simulation and the participating experts were professionals working full-time for a simulation vendor, each of whom had at least 5 years of professional experience.

The level of “Complexity of question from problem owners” and the “Quality level desired by problem owners” varied from simple and predefined in the first laboratory experiment to complete and open for the model developers in the third laboratory experiment. This was achieved by providing participants with different assignments. In the first laboratory experiment the participants received a valid simulation model that needed to be adjusted for several pre-defined simulation experiments. In the second and third experiment the participants started with an empty model. The assignment for the participants of the second laboratory experiments was to design a simulation model that would be usable for experimentation and in the third laboratory experiment the participants had to design an optimal layout for the rail network.

The '+' and '-' signs for the relations in the causal diagram were obtained using the following assumptions, to be confirmed by the laboratory experiments. In the laboratory experiments we will refer to causal relations as described in Table 4.1 and taken from Figure 4.1.

Table 4.1: List of causal relations

ID	Relation	Explanation
1	Experience -> Number of actions	A more experienced model developer will be able to achieve the same result with **fewer** actions.
2	Complexity -> Number of actions	A more complex question will require **more** actions to represent the system in a model.
3	Quality level desired -> Number of actions	A higher level of quality required by the problem owner requires **more** actions of a model developer.
4	Domain specificity -> Number of actions	Instantiating a model using a domain specific extension requires **fewer** actions of the model developers than using a generic environment.
5	Match model constructs -> Number of actions	If the model constructs match the requirements, the model developer does not need to change and adjust the model constructs. If the model constructs do not match, the model developer has to perform **more** actions.
6	Number of actions -> understand model constructs	If more actions are required to develop a model, then this makes it **more** difficult to understand how model constructs in the simulation model represent parts of the system.
7	Number of actions -> understand model structure	If more actions are required to develop a model, then this makes it **more** difficult to understand the model structure.
8	Domain specificity -> understand structure	When instantiating a model using a domain specific extension, it costs **less** effort to understand the structure of the model.
9	Understand model construct -> time spent	Better understanding of the model constructs result in **less** time spent for an experiment
10	Understand structure -> time spent	Better understanding of the simulation model structure results in **less** time spent for an experiment
11	Time spent -> Number questions	Less time spent to carry out an experiment enables the model developer to carry out more experiments and thus answer **more** questions of the problem owner.

4.1.2 Case used in laboratory experiments

The case study applied in the laboratory experiments was based on a master thesis project at Delft University of Technology by Brandt (1999). This master project concerned the analysis of possible vehicles, cost calculations, design of the route and analysis of logistical performances using simulation models. The case study is described in the block text below in the same way it was described to all participants in the three laboratory experiments. This introductory description was extended with information and specific questions that were different for each of the three laboratory experiments. In the first laboratory experiment the questions of the problem owners were formulated in the form of precisely defined simulation experiments that the participants had to perform. In the second laboratory experiment the participants had to compose a valid simulation model that represented the 2010 situation and a fixed timetable for the vehicles. Participants in the third laboratory experiment received the assignment to come up with an optimal solution for the situations in 2015 and 2020 given three possible vehicle vendors.

Over the last years new metropolitan areas have evolved in the area between The Hague Central Station (CS) and the VINEX location Ypenburg. A completely new 'city' has been developed in Ypenburg and in the area between the highway and the Central Station a lot of new office space has been created. Due to these building activities the throughput levels on the existing transport infrastructure have increased and the result is that traffic jams occur each morning and evening and parking a car is becoming increasingly challenging. An alternative to the car can be provided by high quality automated mass-transportation systems. The policy makers of The Hague looked at various alternatives for automatic transportation by monorail which they liked.
In 1999 the SkyShuttle project started to investigate the added value and necessary investment required for a high quality mass transport system. This system should be equipped to handle 15,000 people per a day of which 70% will travel during peak hours. The consultant companies Advanced Netherlands Transport (ANT) and Bohemen Beheer BV have designed some routes. The map of the area with the most likely route, according to the involved transportation experts is shown in Figure 4.2. The yellow stations will be the first to be implemented, the red stations are future extensions expected to be completed in 2015 and 2020.
At these different stations the number of passengers fluctuates over time. In the morning the main flows will go from Central Station to Brinkhorst (the major office centre) and from Ypenburg to either Central Station or to Brinkhorst. In the evening the main passenger flows will be in the opposite direction. Besides commuter traffic, leisure related traffic to HTS and GAVI[4] and business related traffic to the Brinkhorst is expected to use the same infrastructure.
The rough calculations performed by the SkyShuttle project team showed possible benefits. The next step identified by the project team was to provide some additional sources of information regarding the feasibility of the monorail concept. They identified simulation as the methodology that should be used to do this, as simulation can be used to model logistic flows and to visualize them.

[4] HTS = Shopping mall ; GAVI = new soccer stadium of ADO Den Haag

The project team wants you to carry out a very short simulation study to provide some data and a good visualization. Unfortunately due to competitors in the project, bad planning and some other excuses, the simulation study is required to provide some results fast.
The map of the expected route between The Hague and Ypenburg is shown in Figure 4.2. A completely new and advanced automatic transportation system is dealt with in this case study. Although the route is fixed, a lot of design choices are still open and these need to be evaluated using simulation. Some of the choices are:

- type of vehicle, monorail versus cable mover, large versus small vehicles, fast versus slow
- number of vehicles
- daily pattern of the vehicles
- number of platforms at stations
- number of tracks between stations

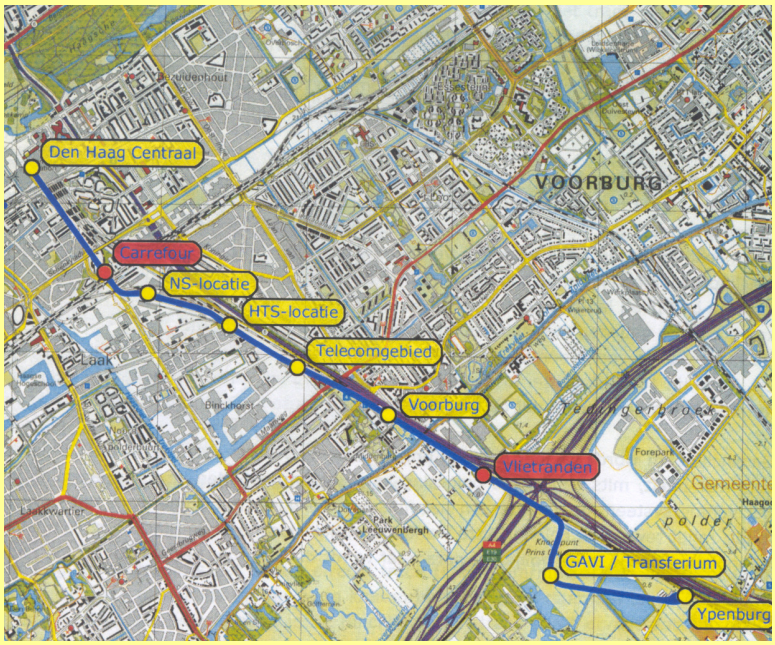


Figure 4.2: Expected route for the SkyShuttle transportation system

All these choices need to be evaluated for a large range of assumptions, one sub-set of assumptions that need to be varied to give a complete overview of the situation is: arrival pattern of passengers, origin-destination of passengers, effects of new offices on leisure activities in the region, effects of linking the new system to present public transportation (bus and train) systems.

4.1.3 Domain specific extension for rail networks

No domain specific extension was available to implement monorail / light rail networks and to observe performance indicators for individual passengers. Therefore a new simulation environment was developed for passengers in rail networks, based on the problem description of Brandt (1999) and simulation studies of rail networks for passengers (Hooghiemstra and Teunisse, 1998).

The systems that were evaluated in the simulation studies of Brandt (1998) and Hooghiemstra and Teunisse (1998) were decomposed, as described in section 2.4, using the object oriented decomposition and process oriented

decomposition. The object oriented decomposition resulted in the system elements station, platform, track, vehicle and passenger. The process oriented decomposition provided process flows for the entities vehicle and passenger. The identified system elements were all translated into domain specific model constructs. These domain specific model constructs have been composed using generic model constructs of the simulation environment Arena.

Model constructs to represent infrastructure

The physical network consists of three model constructs; track, station and platform. The configuration of these model constructs can be retrieved directly from the drawing of the system. Changing the configuration is not difficult thanks to the domain specific interface of the model constructs; an example is shown in Figure 4.3. A simulation model of an example network using the infrastructure model constructs of the domain specific extension is given in Figure 4.4.

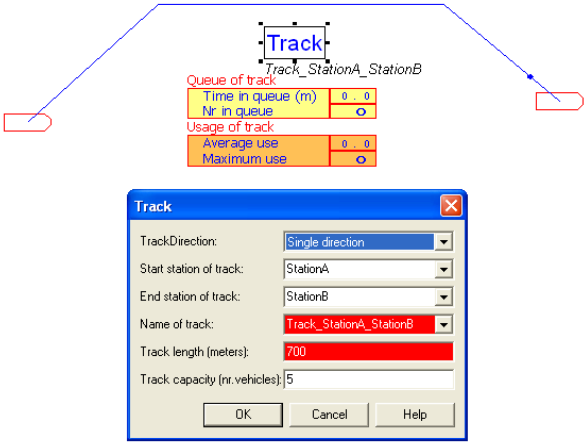


**Figure 4.3: Model construct "Track"
with visualization and interface for parameters**

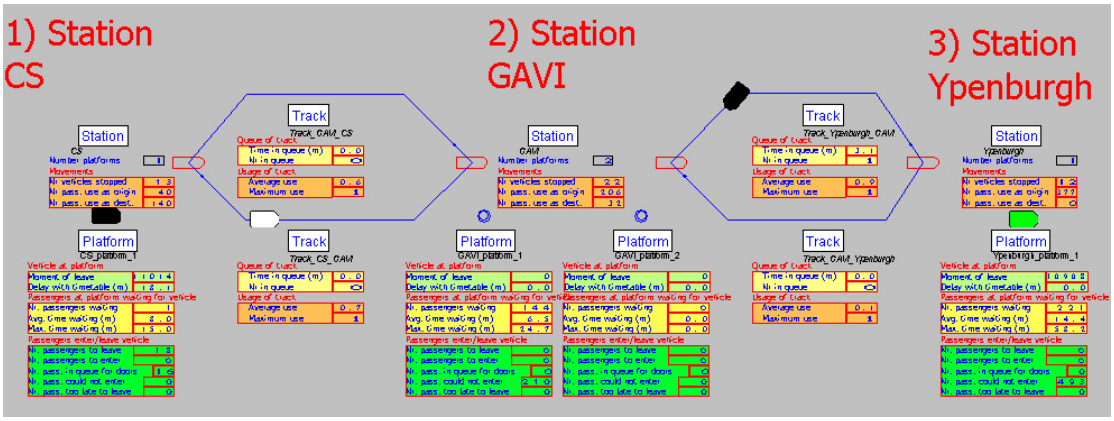


Figure 4.4: Screen dump of the physical network in a simulation model

The model construct “Station” is an implementation of a connection between two or more tracks. The “Station” provides one or more platforms where passengers can get into or out of vehicles. Several processes take place at the platform. Among these processes are ‘passenger waits for a vehicle’, ‘passenger enters vehicle through a door, ‘vehicle stops and leaves’ and ‘vehicle enables passengers to leave’ when they arrive at the platform of their destination station.

The model construct “Track” connects the stations in the network. Tracks can be of the type single or dual direction. Single direction means that vehicles can move only from one start station to the end of the track. Dual direction means that a security system is included so that vehicles can move in both directions along a track. The network in Figure 4.4 shows single tracks between all instances of the station model construct.

Model constructs to control passenger and vehicle entities

The two types of entity, ‘passenger’ and ‘vehicle’, have their own control mechanisms. These control mechanisms describe the process of the entities and their use of the infrastructure, i.e. tracks and stations. The process of the passenger is represented in Figure 4.5 and instantiated in the model construct “PassengerControl”.

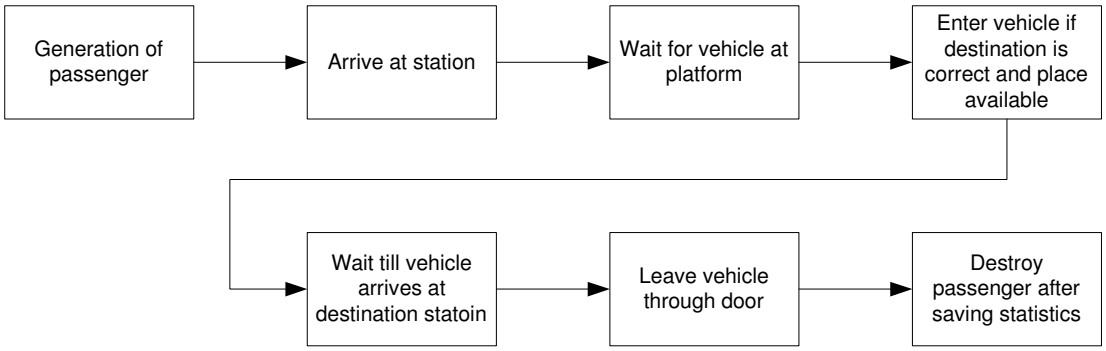


Figure 4.5: Abstract representation of the process of passenger entity

The process of the vehicle entity is shown in Figure 4.6. This process is implemented and managed by the model construct “VehicleControl”. The figure shows only the main process steps.

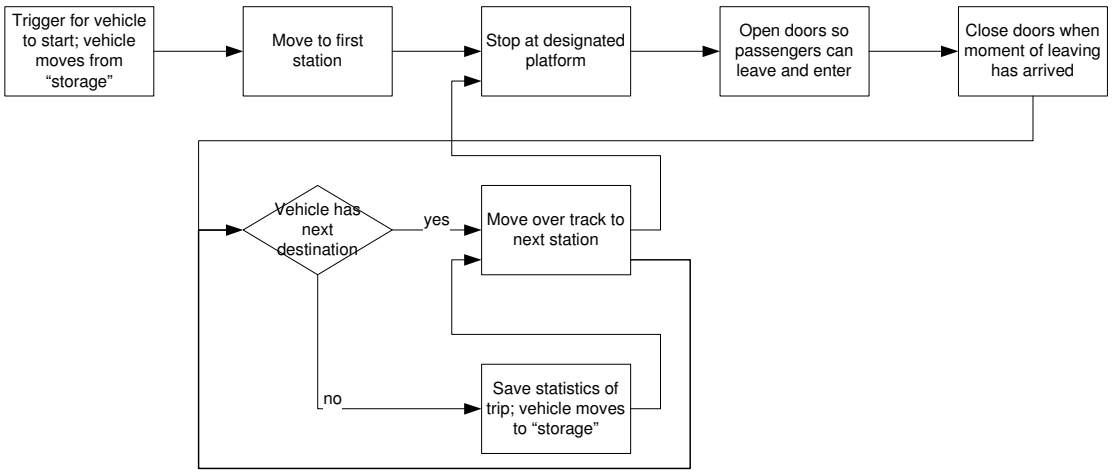


Figure 4.6: Abstract representation of the process of vehicle entity

Validation of the domain specific extension

The model constructs of the domain specific extension were verified and validated by instantiating several small simulation models of imaginary situations and one simulation model of a real problem system. The small simulation models showed that the model constructs worked as expected. Passengers reached their final destinations and vehicles waited for other vehicles to clear platforms and pick up available passengers (Valentin et al, 2003a).

The real-life simulation model represented the same system Brandt (1999) studied. The same input parameters were used to validate the behavior of all the model constructs in a model. Brandt identified the number of travelled kilometers of vehicles and passengers and utilization of vehicles as main performance indicators. The simulation model developed using the domain specific extension for rail networks showed the same values with a 95% confidence interval for these performance indicators.

4.2 First laboratory experiment: experimenting with an existing simulation model

4.2.1 Set up of the first laboratory experiment

The first of the three laboratory experiments can be seen as a follow up of the initial study performed by Brandt (1999). Simulation experts had already developed simulation models, using the domain specific and generic simulation environment, and the participants had to perform different simulation experiments using these models. The required list of experiments was predefined by the problem owner. The simulation study will be effective for the problem owner if all desired experiments are performed. The proposition for this laboratory experiment is:

Simulation novices who use a simulation model composed of domain specific model constructs perform more simulation experiments than simulation novices who use a simulation model composed of model constructs of a generic simulation environment.

In Figure 4.7 circles are drawn around the main items in the causal diagram that are evaluated during the first laboratory experiment. The participants have to understand the simulation model, i.e. the structure and the interface, and with an understanding of the model they can answer questions from the problem owners that deal with either the type of vehicles or with the infrastructure of stations and tracks. The participants were allocated to the domain specific extension or the generic simulation environment at random.

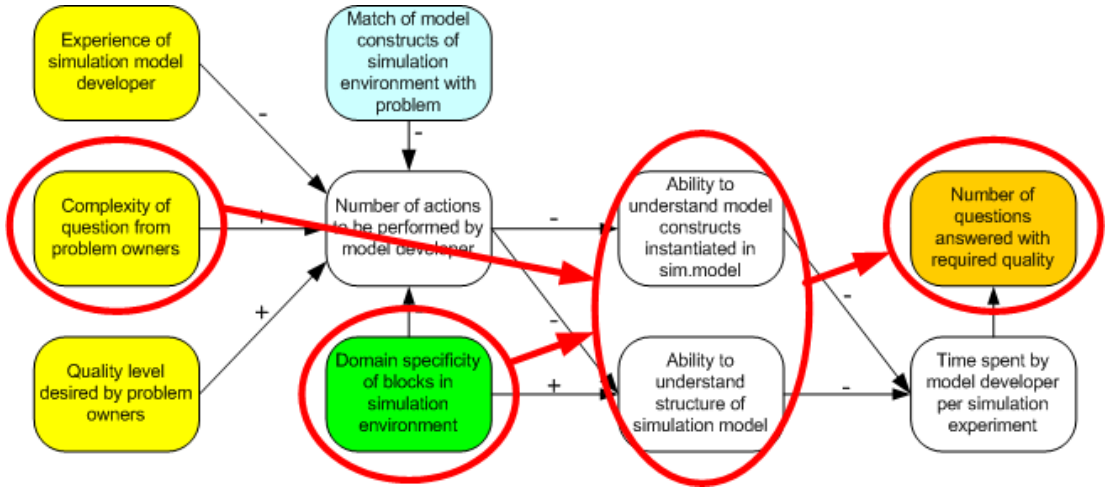


Figure 4.7: Focus of the first laboratory experiment

The participants of this laboratory experiment were 30 full-time students from the Faculty of Technology, Policy and Management of Delft University of Technology in their third and final year of their bachelor degree, or in the first year of their master degree. All these students had received a basic education in the use of Arena (a 160 hours course) and they had carried out a small simulation project (80 hours course). With only educational experience these participants can be called novices in simulation modeling. The laboratory experiment took 6 hours, including preparations and filling in the final questionnaire.

The group of participants was randomly divided into two groups. Participants of one group used simulation models that were constructed using model constructs of the domain specific extension for rail networks. The second group used simulation models that were constructed using model constructs of the generic simulation environment Arena. Neutral observers evaluated both models to check whether the simulation models were good and understandable (Valentin et al, 2003a).

The participants received documentation about how the simulation models worked and where changes could be made. This was described in two different documents, because the individual participants received specific support documentation based on the simulation environment they used.

All participants received the same set of requests for additional simulation experiments after approximately two hours of learning how the simulation model worked (Table 4.2). The last two columns in the table are performance indicators of the simulation model. The participants of the laboratory experiment had to fill in these performance indicates after they finished the simulation experiment. The first assignment consisted of 15 different simulation experiments all concerning adjustments to vehicle behavior. The second assignment consisted of 15 simulation experiments that focused on adjustments to the infrastructure.

Table 4.2: Part of the 15 simulation experiments to be performed by laboratory participants for the first set of assignments

Description	Average waiting	Max. delay vehicle
Change the type of vehicle from Hovair to the monorail of Siemens with one carriage (similar intervals for starting from CS, different timetable)		
Change the type of vehicle from Hovair to the monorail of Siemens with two carriages (similar intervals for starting from CS, different timetable)		
The Hovair vehicle will move each 10 minutes, in addition extra vehicles will be used during peak hours. Morning peak hours are from 8 till 10, afternoon peak hours are from 17 till 19. These extra vehicles will consist of two carriages and move each 5 minutes.		

The participants finished their laboratory experiment by filling in a questionnaire that was used to evaluate their level of satisfaction. The questionnaire was also used to observe the other relations of the causal diagram (Figure 4.1).

4.2.2 Insights gathered, based on output of the first laboratory experiment

Observations during the experiment

A separate modeling assignment based on the Arena modeling exam of 2001 showed that the 30 participants of the first laboratory experiment could be divided into two types of participants. 22 participants scored good, between 25 and 35 points out of 40 available points for the assignment, and 8 participants scored low, between 15 and 25 points. Participants were assigned randomly to work with the domain specific or generic simulation environment, taking into consideration the score of the modeling assignment to ensure even spread of high and low performers.

The participants started with 2 hours of learning to use simulation models of public transportation in their simulation environment. The individuals working with the generic simulation environment used the opportunity to ask a lot of questions regarding the simulation model. These questions covered topics such as how to make certain adjustments, what the meaning was of various things in the simulation model, and why certain model constructs in the generic simulation environment were used. The participants using the simulation model developed in the domain specific extension asked only a couple of questions. They read the material and started immediately to make step-by-step adjustments to the example simulation models using the example assignment.

The participants were all very motivated. They behaved as if they were competing as groups to use the alternative systems. As a result all students worked as hard and well as possible to tackle as many possible experiments even though the participants with the generic simulation environment soon noticed that the participants with domain specific extension moved faster through the experiments.

Prescribed simulation experiments

The first set of simulation experiments that the participants received dealt with the selection of the kind of vehicles to be used on the transport system. This meant that the participants had to perform simulation experiments with more vehicles, larger vehicles or faster vehicles. The changes they had to make to develop simulation models that could be used for these simulation experiments were comparable to the changes they made during the first example assignment. The participants could easily repeat their actions, because they recognized the steps they needed to do to adjust the simulation model for the experiments.

The forms where participants logged the output of their experiment showed that it did not matter for the participants with good Arena skills whether they used the domain specific or the generic simulation environment for these first set of assignments. The participants with good modeling skills succeeded in running between 9 and 12 simulation experiments with valid results. This comparable number of simulation experiments can be explained by the small number of actions the participants needed to carry out for changing the simulation models. The experiments to be performed were also highly repetitive, for 15 simulation experiments two or three very similar adjustments to the simulation model needed to be made.

The use of the domain specific or generic simulation environment showed a clearer difference for the participants with low modeling skills. The participants with low modeling skills who had to use the domain specific extension performed 6 to 10 valid simulation experiments. The other

participants with low modeling skills using the basic simulation environment performed no more than 3 valid simulation experiments, most none at all. The reason was that these participants could not find where to make the changes and therefore began looking at the detailed logic of the simulation model. Evaluating the logic of the simulation model made it difficult for the participants to find the parameters required to adjust. The participants using a simulation model in the domain specific extension did not have to understand the detailed logic inside the domain specific model constructs. They could adjust the parameters in the user-interfaces of the model constructs.

In the second set of simulation experiments the participants had to adjust the infrastructure, i.e. add extra stations with platforms and tracks. The participant had 2 hours for several simulation experiments regarding alternative layouts of tracks and stations. After 2 hours the difference between generic and domain specific was clear. The participants with the generic simulation environment were still making changes to the simulation model for the first simulation experiment. The participants with the domain specific extension had performed 4 to 8 valid simulation experiments. The number of valid performed experiments was not as high as in the first set of experiments, because they made several mistakes while changing their models:

- they carried out no verification or validation after they made the changes to the simulation model and thus they did not observe deadlocks of waiting trains;
- they forgot to enter some data values. The adjustments that had to be made for the second set of simulation experiments concerned the model constructs of infrastructure and control, because a new station implied a new timetable for the vehicles.

The participants using the domain specific extension still had sufficient time to adjust their simulation models and to correct their mistakes once they realized the shortcomings of their first experiments.

Observations based on the questionnaire

The participants finalized their activities for the laboratory experiment by filling in a questionnaire to evaluate whether they felt satisfied with the correctness and validity of their simulation results. The participants had to score 7 statements on a scale from 1 to 5. A score of 1 indicated that they completely disagreed with a statement while a score of 5 indicated that they completely agreed.

We can assume that the participants of the laboratory experiments were a random sample of the group of inexperienced simulation users. The participants were randomly allocated either to use the domain specific extension or the generic simulation environment. As a result, we are allowed to combine the variances of the answers of the questionnaire of the two

groups to one value (McClave et al, 2000). With the combined variance of the sample we can apply a t-test with 28 degrees of freedom.

The outcome of the questionnaire for the two different groups is shown in Table 4.3. In this table no distinction is made for type of participant regarding skill level. The columns “mean” refer to the mean value the participants of domain specific or generic provided. The variances are the variances of the set of answers provided by the participants for domain specific or generic. The value is bold if the t-test (99% certainty, 28 degrees of freedom) showed that the participants using the domain specific extension agreed more with the item in the questionnaire than the participants using the generic simulation environment.

Table 4.3: Outcome of the first laboratory experiment

	Description	**Mean**		**Variances**		$T_{0.01}$ 28df
	N domain specific = 16 ; N generic = 14 1 = completely disagree; 5 = completely agree	**spec -ific**	**gen- eric**	**spec -ific**	**gen- eric**	
1	The adjusted simulation models are technically correct representations of the system	4.1	3.5	0.5	0.6	2.2
2	The adjusted simulation models are valid representations of the system	3.9	3.4	0.5	0.5	1.9
3	Use of the other simulation environment would have resulted in a lower number of executed experiments	3.9	1.9	2.3	0.8	**4.3**
4	The used simulation environment is suitable for modeling a rail network	4.4	2.1	0.2	1.2	**7.5**
5	The simulation model developed in the used simulation environment was easy to understand	4.3	2.0	0.6	0.8	**7.4**
6	The simulation model developed in the used simulation environment was easy to maintain	3.9	1.9	0.5	0.6	**7.3**
7	The simulation model developed in the used simulation environment was easy to extend	3.7	1.6	0.6	0.8	**6.8**

Both groups of participants agreed with the first and second statement. These findings are not surprising given that we observed that the participants were working hard and were proud of the work they had carried out.

The participants only worked with one of the two simulation environments, but they could see what the others were doing, they heard each other’s questions and during the lunch they chatted together and compared their progress. Based on this informal exchange of information and ideas the participants had a perception about the applicability of the other simulation environment for the assignments used in this laboratory experiment. Statements 3 and 4 show that the participants with the domain specific extension preferred their environment above the generic simulation environment and the participants with the generic simulation environments envied the others, supported by the t-values of 4.3 and 7.5.

Most likely the preference for the simulation environment is partly based on the usability, maintainability and extendibility of the simulation model for experimentation. A significant difference can be observed between the participants given the domain specific extensions to use and those given generic simulation environments. Understanding the simulation model of the domain specific extension was much easier than understanding the simulation model based on the generic simulation environment according to statement 5. Increased understandability resulted in better maintainability and better maintainability enabled easy extendibility of the simulation model for the experiments.

4.2.3 Conclusions from first laboratory experiment

Based on the analysis of the observations made during the laboratory experiment and the questionnaire results, we can conclude that performing simulation experiments using a simulation model developed with domain specific model constructs is easier than using a simulation model based on a generic simulation environment (Kolfschoten et al, 2006). The observations of the laboratory experiment show that this qualified advantage is achieved as a result of perceived higher understandability, easier maintainability and extendibility of the simulation model. This enabled the participants with the domain specific extensions to ask for less support, as is seen by the fact that there were almost no questions from these participants, and to work faster, and thus perform relatively more simulation experiments. Further, the understandability of the domain specific extension enabled participants with low modeling skills to still be able to perform a number of simulation experiments and provide insight to problem owners.

Some risks of using domain specific extensions were also encountered. The novices had too much trust and faith in the simulation model developed with the domain specific extension, so they forgot important steps of a simulation study, mainly verification and validation of the model.

Participants that used the simulation model developed in the domain specific extension carried out more valid simulation experiments, mainly in the second set of assignments. This view is strengthened by the results of the questionnaire and thus the proposition of this laboratory experiment can be accepted:

> *Simulation novices that use a simulation model composed of domain specific model constructs perform more simulation experiments than simulation novices that use a simulation model composed of model constructs of a generic simulation environment.*

From the causal diagram on page 92 one can see that the first laboratory experiment focuses on the complexity of the questions, the domain specificity and the effects with respect to understanding the simulation model. The questionnaire confirmed that domain specificity reduces the effort required to understand the simulation model (causal relation 8). This causal relation is thus negative as expected. The observations showed that more complex questions result in more actions (causal relation 2) and that the larger number of actions that needed to be carried out for the second set of laboratory experiments made the model understanding less (causal relations 6 and 7). The importance of the understanding is also observed in the differences between the first set of experiments, i.e. experiments for vehicle configuration, and the second set of experiments, i.e. new stations. In the second set of experiments more actions had to be carried out resulting in less understanding of the model constructs and structure (causal relations 6 and 7).

4.3 Second laboratory experiment: creating simulation models from scratch

4.3.1 Set up of the second laboratory experiment

In the second laboratory experiment participants were asked to develop a full simulation model from scratch. The same problem situation and simulation environments were provided for the participants of this laboratory experiment. The insight that the problem owner required was defined as open ended. The participants did not have to perform a fixed set of experiments or deliver a certain animation of the system in operation. The participants had to make an educated guess as to what information would be sufficient for the problem owner.

The participants had the task to develop a valid simulation model within 8 hours. In this laboratory experiment it was key to finish in time. After these 8 hours an evaluation was done on how much insight the model developer provided to the problem owner and whether that covered all the items the problem owner requested. Possibly participants would be finished in less time and be able to provide the insight for the problem owner earlier. Therefore effectiveness was interpreted as “better and faster”, resulting in the following proposition for the second laboratory experiment:

> *Simulation novices can develop a simulation model better and faster using model constructs from a domain specific extension for future simulation experiments than simulation novices that must develop a simulation model composed of model constructs from a generic simulation environment.*

'Faster' was evaluated by comparing the moments the students claimed to be ready or, if they had not finished in 8 hours, by the participants making an educated guess regarding how much more time was needed. 'Better' was evaluated by observing the ability of the simulation models made by the participants to provide insight into the questions of the SkyShuttle problem owners (see the case description on page 87).

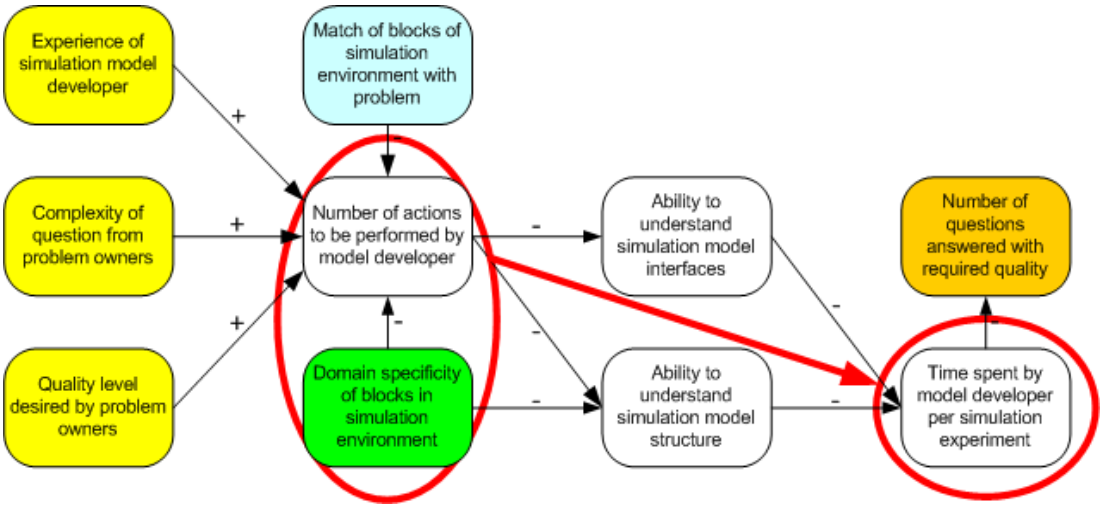


Figure 4.8: Focus of the second laboratory experiment

The participants had only one question to answer, but depending on the domain specificity they have to carry out a number of actions. The main measurement in this laboratory experiment was whether the simulation novices succeeded in building a valid model and performing a simulation experiment. Once the model developers could perform a valid simulation experiment they were done, therefore in Figure 4.8 a circle is drawn around the time spent and not around the number of questions answered.

This laboratory experiment was done by 16 students of the Faculty of Technology, Policy and Management of Delft University of Technology. All the students had received a basic education comparable to the participants in the first laboratory experiment. In addition, the students had been trained for 12 weeks in advanced simulation model development.

The participants randomly received the domain specific or generic simulation environment. All the participants received documentation about the concepts that could be used for building the simulation models. These concepts formed the basis of the model constructs from the domain specific extension, but it was made clear to all participants that the conceptual model was just an overview and they could reduce or expand the model concepts as much as they felt appropriate. The individual participants using model constructs from the domain specific extension received some extra material about the background and technical implementation of the available model

constructs. The time that the participants needed to study this material was included as part of the 8 hours allowed for the complete laboratory experiment.

At the end of the laboratory experiment the students were required to provide a set of deliverables:

- a simulation model based on the provided simulation environment,
- a document that explained the chosen level of abstraction and the boundaries used for the simulation models,
- a filled in questionnaire regarding their satisfaction with using the assigned simulation environment and their expectations of the model development assignment,
- a log-file registering their activities of the 8 hours.

Two simulation experts with several years of experience evaluated the models and documents, mainly the simulation model and the explanatory document (Valentin et al, 2003b). These experts were expected to judge the simulation model on clarity, structure, ease of maintenance and ease of extension. Further they were required to judge the completeness of the simulation model and to see whether the participants had implemented their assumptions that were described in the explanatory document.

The expectation at the beginning of the second laboratory experiment was that the proposition at the beginning of this section would be accepted. This expectation was based on the following additional expected observations:

- the simulation models based on the model constructs of the domain specific extension will contain more details than the models of the generic simulation environment,
- the participants using the domain specific extension will be more positive about the quality of their simulation models, compared to the model developers using the generic simulation environment,
- the participants using the domain specific extension will assume they have better met the problem owner's needs regarding visualization, performance indicators and preparation for future experiments, compared to the model developers using the generic simulation environment,
- the participants using the domain specific extension will agree more with the statement that they have had enough time compared to the model developers using the generic simulation environment.

4.3.2 Insight gathered, based on the second laboratory experiment

The participants made less progress with the simulation study than expected (Valentin et al, 2003a). Only one participant claimed to have finished. He provided a simulation model and several simulation experiments that showed the validity of the model within the period of 8 hours. All the other participants ran out of time for the laboratory experiment before they were finished. Therefore, the original plan to evaluate the final simulation models and judge their quality could not be carried out as planned. The time log could not be used either, because the participants logged that they only carried out the task “model development”. Nevertheless the simulation models handed in were analyzed, but we focused more on what was lacking rather than on what accomplished. Below, a description is given, per type of observation of what was learned from this laboratory experiment.

Observation during the performance of the experiment

The participants using model constructs of the domain specific extension started with reading the provided documents to help them understand the functionality of the available model constructs. Once they felt confident about their knowledge regarding how to use the model constructs, the participants started to develop a simulation model. Their actions consisted mainly of quickly instantiating their model using the model constructs, and then entering data for all the parameters of the model constructs. The participants immediately tried to run their simulation model after they had instantiated the complete simulation model using all the available data. This showed that they had faith in the model constructs of the simulation environment and trusted the model constructs to work flawless. Unfortunately, the participants received error messages relating to typing errors and missing instances of model constructs. Solving these error messages took them a large amount of time.

The participants using the domain specific extension started to validate the simulation model, when their model was finally compiled and seemed to run. Validating the simulation model took them more time than they expected, mainly because the data for passenger arrival lead to a highly unstable system, which meant that selecting vehicle types and the timetable was a hard task. Some of the students carried out as many as 15 different simulation experiments just to have an idea of whether their simulation run was valid.

When trying to solve the modeling errors the participants made verbal remarks to express their frustration and to keep up their spirits. Some of their remarks are given below. The remarks show that, even though the participants made progress, they did not fully understand what was happening.

- *"This is nice, something is moving. I do not know what, but it is moving".*
- *"I have defined that vehicles stop for 2 minutes, but what the **** is 'minimum stop time'? I do not understand anything of these model constructs".*
- *"The fast vehicles are running over the slow vehicles on the same track".*

The participants who used the model constructs of the generic simulation environment started with refreshing their knowledge on suitable model constructs of the simulation environment. They frequently used the help-files and basic modeling examples provided by the generic simulation environment. Based on these small training models they decided how to use the generic model constructs, but their lack of a broad modeling experience prevented them from making fast progress in model development. Each new functionality they added to the simulation model was tested and made valid, to make sure that they would not have the problem of composing a complete simulation model and being incapable solving errors from the simulation environment.

The approach of the participants using the generic simulation environment seemed good, but some quotes show that they were surprised that the participants using domain specific extensions were much faster:

- participant with domain specific: *"How can you increase the speed of your run?"* participant with generic: *"Are you already finished then?"*
- participant with domain specific: "*I have a problem, I would like my animated vehicles to move exactly following the line*" participant with generic: *"That's your problem? My model is not running and I do not expect it to run at all"*
- *"I have reduced the system so that passengers only move from left to right"*
- *"I have reduced a lot already, but I should have reduced it so that I have no more than one station in the system"*

Observations made by the simulation experts

The two simulation experts who had to evaluate the simulation models and the provided documents of the participants had a reasonably simple job. They looked at the models and judged whether the models were complete, i.e. that they contained a minimum set of functionalities at a reasonable level of detail. The simulation experts took into account the documentation of the participants that described what kind of abstraction and limitations they had applied.

The conclusion of the simulation experts was that the simulation models of the participants using model constructs from the domain specific extension were almost finished and that these participants were already working on the experimentation.

The participants who used the model constructs of the generic simulation environment Arena had not progressed as far as the participants using domain specific extension within their 8-hour time frame according to the simulation experts. Based on the simulation models that the participants made it was concluded that they used a very structured approach, starting with a small simulation model and extending it with new functionalities. The final simulation models included one or more moving vehicles and a first attempt at passenger generation. In the documentation the participants stated that they were almost finished, but the observing simulation experts judged that the participants would need to do more to enable all required experiments to be performed with the model.

Observations based on the questionnaire

The questionnaire filled in by the participants of the second laboratory experiment has a limited value. The number of participants in the experiment was small and thus the number of respondents to the questionnaire was also low. Table 4.4 summarizes a few of the most important statements and the average scored opinion of the participants. Even though there was a small number of participants, we could carry out a t-test for comparing the two groups, using 14 degrees of freedom.

The outcome of the questionnaire for the two different groups is shown in Table 4.4. In this table no distinction is made for type of participant regarding skill level. The columns “mean” refer to the mean value the participants of domain specific or generic provided. The variances are the variances of the set of answers provided by the participants for domain specific or generic. The t-value was calculated using the variance of the complete sample, weighted for 9 domain specific participants and 7 generic participants. The value is bold if the t-test showed that the participants using the domain specific extension agreed more with the statement than the participants using the generic simulation environment or the other way around.

Statements one to five in the questionnaire dealt with satisfaction of the participants with the simulation model they developed. The average scores of the participants using model constructs from the domain specific extension for these statements ranged from 2.4 to 3.1 on a 5 point scale. This means that on average the participants were not satisfied with the quality of their work. These participants thought that the validation of their simulation model and the availability of performance indicators could be improved. The average scores of the participants using model constructs from the generic simulation

environment were even lower for these five statements. The average score of these participants is not higher than 2.0, meaning that the participants were dissatisfied with the model development, validity, and visualization.

The sixth statement in Table 4.4 shows that the participants using the generic simulation environment unanimously agreed that there was not enough time to develop the simulation model. Some of the participants using the domain specific extension felt that 8 hours was enough for this task.

Statements 7, 8 and 9 provide information about the understandability and future adjustability of the simulation model. The participants using the domain specific extension again agreed more on this than the participants using the generic simulation environments. The latter assume that problem owners can understand their model reasonably well, but they see extendibility and maintainability as a problem.

Table 4.4: Outcome of the second laboratory experiment

	Description		**Mean**		**Variances**		$T_{0.01}$ 14df
	N domain specific = 9 ; N generic = 7 1 = completely disagree; 5 = completely agree		**spec -ific**	**gen- eric**	**spec -ific**	**gen- eric**	
1	Satisfaction of model developers about the simulation model	The simulation model is technically correct / verified	3.4	1.9	0.6	0.8	**3.5**
2		The simulation model is a valid representation of the system	2.4	1.3	0.3	0.1	**5.5**
3		The used level of abstraction was correct to answer the questions of the problem owners	3.0	2.0	0.5	0.4	**3.0**
4		The visualization of the simulation model meets the problem owner's needs	3.1	1.0	0.8	0.0	**6.1**
5		The performance indicators of the simulation model meet the problem owner's needs	2.8	1.1	0.6	0.4	**4.7**
6	Time	Not enough time was provided for model development	2.9	4.9	1	0.1	**-5.9**
7	Future use of the simulation model	The simulation models developed are easy to understand by problem owners	3.1	3.0	0.7	1.0	0.2
8		The simulation models developed are easy to maintain for later simulation studies	3.4	1.0	0.5	0.1	**8.7**
9		The simulation models developed are easy to extend for later simulation studies	3.2	1.4	0.4	0.1	**7.2**

4.3.3 Conclusions from the second laboratory experiment

This laboratory experiment showed that, for novice simulation model developers, the development of a simulation model using model constructs of a domain specific extension is easier and faster than the development of a

similar simulation model using the model constructs of a generic simulation environment. The participants using domain specific extensions achieved more, built better simulation models, were more confident about their model, and carried out more evaluations to improve the quality of their simulation model within the available 8 hours. Thus the proposition for this laboratory experiment (see page 98) can not be rejected.

Even though the participants using the domain specific extension achieved better results, it was interesting how they reached their results. Their approach of: “first we implement everything and enter all the data; then we test” showed a high confidence in the power and user-friendliness of the domain specific extension. They expected error messages during the development process to tell them that they were making mistakes. This confidence cost these participants a lot of time as they had to deal later with the mistakes they made. Support from model development and a list of frequently made mistakes in addition to the available documentation, might have helped these participants.

The participants who were using the domain specific extension got closer to completion of the simulation model. The causal relation between domain specificity and number of actions (causal relation 4 in Table 4.1 and Figure 4.1) showed that we expected that fewer actions should be carried out. This relation turned out to be true for model development. The participants that were using the domain specific experiments needed much fewer steps to instantiate their first simulation model. However, these participants used a lot of time to check the simulation model and to try to validate it completely. Even though fewer actions were required, the understanding of the model constructs in the simulation model was not better (causal relation 6 in Table 4.1 and Figure 4.1). Only after the participants had gathered sufficient understanding of the simulation model, the simulation experiment could be completed.

4.4 Third laboratory experiment: performing a simulation study

4.4.1 Set up of the third laboratory experiment

The participants in the first and second laboratory experiment were novices. None of them had extensive experience in developing a simulation model or performing a simulation study. Their lack of experience with model reduction and their lack of knowledge of the generic simulation environment used, resulted in difficulties with implementing and recognizing the error messages of the simulation environment. Experience with using simulation models and a generic simulation environment may affect how a model developer uses a domain specific extension. Professional simulation experts with a good working knowledge of the generic simulation environment Arena performed the third laboratory experiment. The assignment for the experts

was to carry out a complete simulation study in which the simulation model had to be built from scratch, using either the domain specific or the generic simulation environment. The quality of the simulation model was evaluated by showing the output of the case study to real problem owners.

Eight employees from the company Rockwell Software, the vendor of the generic simulation environment Arena that was used in the laboratory experiments, participated in the experiment. All of the participating simulation experts had been working for at least 5 years in simulation, some of them for 20 years. The group consisted of developers of the core code of the simulation environment, developers of commercial domain specific extensions, and consultants that use the generic simulation environment in commercial projects. The eight simulation experts were divided into four pairs:

- developers of simulation software with more than 15 years of experience and a PhD in computer science.
- expert users with ± 10 years of experience.
- developers of commercial domain specific extensions as extensions of the generic simulation environment with ± 7 years experience.
- junior simulation consultants with ± 3 years of experience in simulation projects.

Randomly one person of the pair carried out the simulation study with the generic simulation environment. The other person of the pair received the domain specific extension including available documentation and example models. The participants were given a maximum of 8 hours to develop their model and to run any number of experiments they assumed to be necessary. The experiments were meant to demonstrate the validity of the simulation models and to provide the optimal solution for the design of SkyShuttle. None of the participants was able to participate in one session of 8 hours, due to other planned activities, so they divided the 8 hours of the laboratory experiment over a period of three days.

At the end of the laboratory experiment the participants were expected to provide a set of deliverables:

- a simulation model based on the simulation environment assigned.
- a presentation that could be used to provide insight into the optimum system configuration to the problem owners at the Municipality of The Hague.
- a filled in questionnaire about their satisfaction with using the simulation environment assigned to them and their expectations of successful applying the simulation model for (future) problem solving.
- a log-file describing their simulation study activities of the 8-hour period.

The deliverables were important to evaluate the performance of the participants. This evaluation was carried out in three steps. The first step dealt with problem owners in the SkyShuttle project. Problem owners needed to feel supported by the simulation expert, based on model output, visualization, experiments and useful model abstractions. The second step was to judge the quality of the simulation model, on level of detail, completeness, model structure and ease of adjustment using simulation experts. The third step was to evaluate the questionnaire and log files created by the participants.

The project did not have real problem owners to carry out the evaluation, because the case in the laboratory experiment was adapted from the original study of Brandt (1999). However, the problem owners of the initial study that triggered this set-up for the laboratory experiment and 2 experts drawn from the field of transportation who had used simulation models in their projects were willing to participate as problem owners. A list of more than 50 items, that are important to problem owners in the field of transportation, was designed and the problem owners set the priority for these items, leading to 15 items with the highest priority for the problem owners. A similar evaluation was planned for the simulation experts, who were expected to prioritize a large list of items and score the top 15 items for the final simulation models of each of the participants. The lists of criteria for problem owners and simulation experts were developed together with R. Sadowski from Rockwell Software (Valentin and Sadowski, 2003).

It was expected that the evaluations by the problem owners would show that the problem owners gathered more insight from the simulation studies carried out using the domain specific extension. The gap for the requested insight of problem owners would be smaller, and might even disappear completely if the model developers provided all the insight the problem owners requested. The existence of the gap between provided and required insight was defined as the effectiveness of the simulation study. Therefore, the proposition for the third laboratory experiment was:

> *The effectiveness of the described simulation study is larger when simulation experts use model constructs from a domain specific extension than simulation experts that use model constructs from a generic simulation environment.*

This proposition is almost the same as the accepted propositions for the two laboratory experiments using novices. The main difference that was expected between the laboratory experiments with novices and experts was that in this experiment the participants using the domain specific extension would all succeed to build their model within the 8 hour time frame, and would be able to perform several simulation experiments. Secondly, the participants using the generic simulation environment would make their simulation models

at a higher level of abstraction. The similarity between the second and third laboratory experiment result in a focus on the causal relation between the experience of simulation model developers and the number of actions they have to carry out, and as a result the time spent on the simulation study (Figure 4.9).

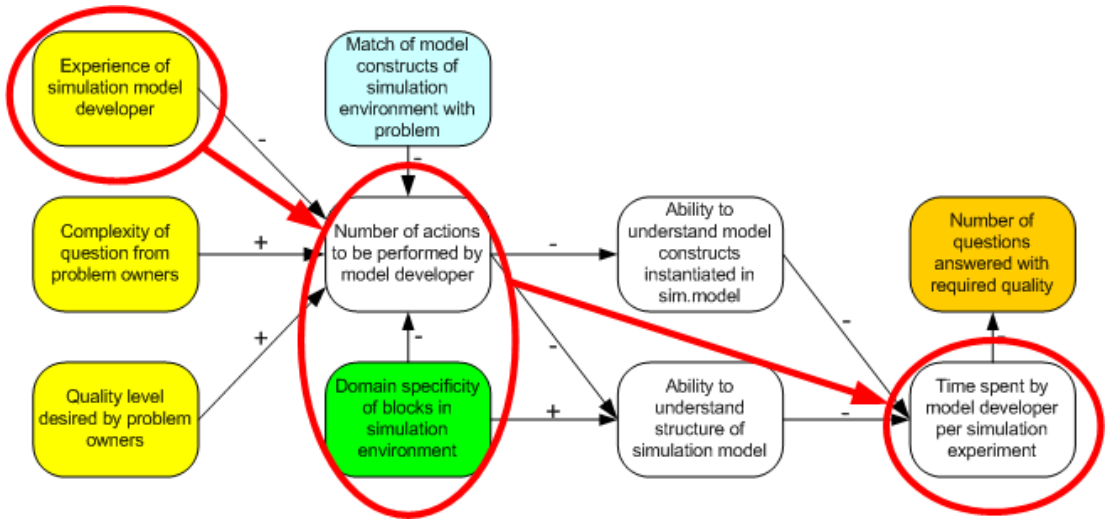


Figure 4.9: Focus of the third laboratory experiment

4.4.2 Insight gathered, based on the third laboratory experiment

It was expected that all the participants would easily finish the complete simulation study. Surprisingly only two participants, both using the model constructs of the domain specific extension, succeeded in performing several simulation experiments and providing output. However, the two simulation experts felt they needed to carry out many more experiments to reach an economically feasible system design and therefore they did not hand in their presentation. As a result the evaluation by problem owners did not take place.

Observations during the laboratory experiment

Notes were made during the laboratory experiment regarding the model building process and the simulation models produced. Even though the participants were all working in their own office, the processes the experts used to carry out the simulation study and the positive and negative results were remarkably similar. The solutions varied in details, but in general the same process was applied by the simulation experts, with similar problems arising during the laboratory experiment.

In the second laboratory experiment, both the experts and the novices used model constructs from the domain specific extension to build their complete simulation model based on the available data. The experts had the same problems as the novices. Error messages and invalid behavior was

observed while they tested their model after the complete model instantiation. However, the way the experts solved their problems differed from those of the novices. The novices mainly went further into the manual and, at random, made adjustments to the data entry trying to find a solution. The novices expected the model constructs to be correct and assumed their data was entered incorrectly. The experts were more convinced about their own work and blamed the model constructs of the domain specific extension. They applied reverse engineering to see where something went wrong in the model constructs of the domain specific extension so they could try to take counter actions using generic model constructs taken from the simulation environment Arena.

- The experts used the debugger of the Arena simulation software and scanned, at the lowest implementation level, the processes carried out within the model constructs of the domain specific extension.
- The experts analyzed the functionality of a model construct by back tracing its internal code. The simulation software allows model developers to see parts of the code of a model construct, even though the simulation experts did not have the source code.
- The experts practiced with example models to understand the process of using the model constructs of the domain specific extension. Testing of the example models using the provided example assignments taught the experts what influences different variables in the user-interface of the model constructs have on the output of the simulation model.

The experts using generic model constructs tried to develop a perfect and broadly applicable simulation model, while the experts using domain specific model constructs struggled with their error messages and deadlocks. The list of possible experiments in the problem description and the provided conceptual models triggered them to try to develop a simulation model that could be easily adjusted to model all the issues mentioned. Unfortunately, time pressure meant that the experts were not able to achieve this level of perfection. Even with their years of experience they made mistakes similar to the novices.

Outcome of the questionnaire and talks after the expert experiment

This laboratory experiment was carried out by eight persons, not sufficient to use statistical techniques to accept or reject the proposition of page 108. In the evaluation of this laboratory experiment the questionnaires were used as triggers for a group evaluation with all the eight participating simulation experts. Table 4.3 shows the outcome of the questionnaire, but no t-test is carried out due to the small number of participants.

All simulation expert participants were satisfied with their work. The participating simulation experts that had been using the domain specific

extension were satisfied that, with their limited knowledge of the domain specific model constructs, they had succeeded in developing a working model and some of them even provided usable model output. In the discussion afterwards it became clear that the experts had more difficulty learning how to work with the domain specific extension than expected from the previous laboratory experiments with the novices. The experts even required more time and material than the novices, as their skepticism regarding the domain specific extensions was much higher than that of the novices. The simulation experts indicated that they would be able to increase productivity and would be able to obtain better results, if they would be using the domain specific extension for a second time. The experts using the generic simulation environment were convinced that they had reached the maximum achievable within the time frame. During the discussion session, the main discussion of these experts was around other approaches such as starting at a higher level of abstraction, reducing data, or ignoring some of the requested experiments.

Table 4.5: Outcome of the third laboratory experiment

	Description		**Mean**		**Variances**	
	N domain specific = 4 ; N generic = 4 1 = completely disagree; 5 = completely agree		**spec-ific**	**gen-eric**	**spec-ific**	**gen-eric**
1	Satisfaction of model developers about the simulation model	The simulation model is technically correct / verified	3.8	1.3	0.5	0.5
2		The simulation model is a valid representation of the system	3.3	1.3	0.5	0.5
3		The used level of abstraction was correct to answer the questions of the problem owners	4.0	3.3	0.8	0.5
4		The visualization of the simulation model meets the problem owner's needs	3.5	1.0	0.6	0.0
5		The performance indicators of the simulation model meet the problem owner's needs	4.3	1.0	0.5	0.0
6	Time	Not enough time was provided for model development	4.8	5.0	0.5	0.0
7	Future use of the simulation model	The simulation models developed are easy to understand by problem owners	3.8	3.0	0.5	0.8
8		The simulation models developed are easy to maintain for later simulation studies	3.5	2.5	0.6	0.6
9		The simulation models developed are easy to extend for later simulation studies	3.8	3.0	0.5	0.0

The answers to the questionnaire showed that all the experts assumed that their simulation model, finished or not, fitted the problem. They were all convinced that the level of abstraction and data was appropriate for providing verified and valid simulation models.

The expected satisfaction of the problem owners of the SkyShuttle project regarding the presented results of the simulation study was scored low by the simulation experts. The participants using the generic simulation environment argued that visualization and performance indicators could be made much better, but they did not have time to do this. The participants using the model constructs from the domain specific extension were slightly more positive about the visualization and performance indicators within their simulation model. None of the participants expected the problem owner to be fully satisfied with the insights provided by the simulation experiments.

The questions in the questionnaires regarding the time limits and use of additional time were not as relevant as expected, because none of the expert participants was close to finishing the project. One of the participants using the domain specific extension claimed "*With the experience I have right now, I could do the project in two hours. I just lost too much time trying to understand what was going on.*" The other users of the domain specific extension agreed. The experts using the domain specific extension admitted that they did not trust the model constructs when the first error messages appeared. The extra work to gather the knowledge they required consumed all the available time, but provided them with the knowledge and trust necessary to use the domain specific extension in future projects.

The experts using the model constructs from the domain specific extension were convinced that their simulation model could be used to provide answers to most of the questions of future problem owners. The experts using the generic simulation environment had the same opinion, but they made the reservation that quite some time would be needed to finish their simulation model and implement the missing functionalities before future problem owners would be satisfied.

Overall the work of the simulation experts using the generic simulation environment confirmed the difficult nature of simulation projects. Even though these experts had years of experience, the expected results were not achieved in time. Too much detail, alternative interpretations of the problem description and an underestimation of the complexity of the process resulted in implementation problems and difficulties in carrying out the required experiments.

The participants using model constructs from the domain specific extension added that the domain specific extension was not as understandable as they expected. The error messages and mysterious deadlocks caused them severe problems. They reacted positively to the idea of providing something like a Frequently-Asked-Question list to help users deal with common problems. The simulation experts also complained that they did not have the source-code, so when they encountered problems, they

could not check the source code of the model constructs of the domain specific extension.

Conclusions of the third laboratory experiment

This laboratory experiment showed that, for simulation experts, developing a simulation model using a domain specific extension is easier and faster than developing a similar simulation model using model constructs from a generic simulation environment. However, simulation experts require higher investments in time and training before they are convinced of the quality of model constructs developed by someone else. The barrier that simulation experts have to overcome before they are confident enough to trust model constructs of domain specific extensions is higher than that for novices. After working with the domain specific extension for several hours the experts gained the necessary trust and confidence. Using this trust and confidence, the simulation experts produced more and better simulation models, they were more confident about their models and they carried out more simulation experiments to improve the system design for the SkyShuttle problem owners within the available 8 hours. Thus the proposition for this laboratory experiment cannot be rejected.

Probably the most important observation of this laboratory experiment is that the experts using model constructs from a domain specific extension had to overcome a high level of mistrust in the model constructs before they could begin to use them comfortably. Simulation experts need to be fully convinced of the technical superiority of the simulation environment and the applicability of the model constructs to the problem before they feel happy to use the domain specific extension. If experts do not fully trust a domain specific extension they will complain about the implementation of the concepts, instead of assuming that they might be mistaken.

The participants using model constructs from the generic simulation environment wanted to show off their expertise with Arena. They were convinced of the quality of their generic simulation skills so they did not want to abstract too much. However, this approach resulted in no solution at all within the time constraints of the experiment.

The causal relation between experts, domain specificity and understandability of the simulation model was unexpected. Experts in simulation models that were working with the generic simulation environment had a high level of understanding of their simulation model. The understanding of the experts using the domain specific extension was much lower. These experts tried to understand, but this took them a lot of additional actions and steps. The number of actions should be divided into the actions required for composing the model and the actions for validating the model. The number of actions required to compose a model using a domain specific

extension is the same for simulation experts and novices, but the number of actions required for validation is much higher for simulation experts, because the simulation experts tend to doubt the validity of the model constructs of the domain specific extension.

Kolfschoten et al (2006) explain that the difficulty that experts have with model constructs is due to the way they handle observations of a system and mentally build their own cognitive scheme. They explain, using cognitive load theory, the advantages of building blocks and the difficulty that (simulation) experts have, because they have to put aside their normal way of working and adapt to the concepts enforced by the model constructs. Novices such as the participants in the first and second laboratory experiment do not have this expertise and thus are more easily able to adapt to the concepts of the domain specific extension.

4.5 Overall conclusions drawn from laboratory experiments

The laboratory experiments showed that model developers using a domain specific extension achieved more results within the given time than model developers that started with a generic simulation environment and the same time limit. ‘More results’ refers to more experiments carried out in laboratory experiment 1 or simulation models that are closer to being used for experimentation in laboratory experiments 2 and 3. The model developers using model constructs from a domain specific extension provided more simulation experiments, delivered simulation models that were better understandable and extendable, and were more satisfied about the usability and quality of their work. Further, the model developers expect that they would need less time to carry out a subsequent simulation study now that they had a good working knowledge of the domain specific extension.

The outcome of the laboratory experiments taught us that the importance of a model developer’s understanding of the model constructs had been underestimated. The different participants in the laboratory experiments taught us that the type of documentation and training given before using a domain specific extension should be tailored to fit the type of simulation model developer. Experts need to be convinced that the model constructs of the domain specific extension are valid before letting them work with a domain specific extension. The main thing that these technical experts wanted, was insight into the inner working of the model constructs of the domain specific extension. The novices, who had fewer prejudices to overcome, could start working directly with the model constructs, but they needed to be managed more within the modeling process. A structured process for using a domain specific extension for developing a simulation model might be very helpful to overcome these problems. Clear support, example solutions in the case of errors, and a Frequently Asked Question list are also needed.

In the laboratory experiments all activities of a simulation study, as described in figure 1.7 were carried out. The laboratory experiments also showed us that the participants experienced most of the advantages and encountered some of the risks that were identified in sections 2.4 and 2.5 for each of the activities. The main differences we observed were for the actions carried out for the verification and validation activity. The novices in the first laboratory experiment were quite optimistic about the quality of their models, so they mainly ignored this activity. The advantage of a simplified verification and validation process resulted in a higher risk of not performing this activity.

The laboratory experiments did not confirm all risks mentioned in literature. The only documented risk encountered during the laboratory experiments was found in the activity “instantiate simulation model” where it turned out that participants did not fully understand the model constructs. This resulted in a longer process to address modeling mistakes.

5 Domain specific extensions realized by simulation building blocks

5.1 Introduction

Chapter 2 described what domain specific extensions are and what steps should be taken to develop domain specific extensions. We then developed new domain specific extensions for automatic guided vehicles and passengers in airports (chapter 3). Additionally, laboratory experiments comparing generic simulation environments and domain specific extensions were carried out in chapter 4. In chapter 3 we observed successful simulation studies due to the flexibility that domain specific extensions offer. The flexibility provided mechanisms to deal with the new requirements and the demands for insight of problem owners and model developers. We also observed that the effectiveness of simulation studies with domain specific extensions improves if we avoid more of the identified risks. In the case studies presented in chapter 3 we still encountered some of the known risks, but we succeeded in overcoming these without any real problems. We should also be able to avoid the new risks we encountered during the case studies in chapter 3 using the knowledge we gained during the case studies. In chapter 4 we learned that domain specific extensions are more effective than generic simulation environments, but that (expert) model developers have difficulties using a new domain specific extension.

The findings in literature and observations from chapter 3 and 4 enabled us to answer the research question introduced in chapter 1 positively:

> *Yes, the effectiveness of a simulation study increases when the simulation models are instantiated using model constructs of a domain specific extension for a simulation environment.*

However, this answer comes with a big reservation. The results of the laboratory experiments and the simulation studies were positive: But during the experiments the developers of the domain specific extension were available to provide support to the participants. We doubt that the simulation studies described in chapter 3 would have finished as successful if the developers of the model constructs had not actively participated and helped the participants during the simulation studies.

The encountered risks were overcome by spending more time and effort on the simulation study. Most of these risks have been known for a long time and have been identified in literature (see chapter 2). The way we carried out the design process of the domain specific model constructs reduced the effect of the risks, but was not sufficient to avoid the risks completely during the simulation study. Time, money and goodwill are lost when carrying out a

simulation study when one needs to mitigate risks. The simulation studies with the domain specific extension were therefore not as effective as they could have been. It will never be possible to create domain specific extensions that are fully satisfactory for a simulation study, but improvements to the concept and design approach of domain specific extensions and the training of model developers will reduce the chance of encountering a risk, and they will support the model developer in mitigating these risks more easily. The things we learned during the case studies that help us to avoid and overcome risks can be translated to requirements for domain specific extensions of simulation studies. These requirements for domain specific extensions will be derived in section 5.2.

The requirements for domain specific extensions cannot be met by just using the concept and design guidelines described in chapter 2. Extensions and changes to the concept and design guidelines need to be made to improve the domain specific extensions. The suggested changes and extensions will be described in section 5.3. A more detailed description of four types of changes and extensions will be provided in the last four sections of this chapter:

- An improved concept of model constructs with structure and interfaces to interact with other model constructs of domain specific extensions (section 5.4).
- Tools and instruments that are part of a domain specific extension to automate the activities the model developers have to perform (section 5.5).
- Training and documentation materials to support a model developer to work independent from the developer of the domain specific extension (section 5.6).
- A design approach to support developers of domain specific extensions to design and implement model according to the new concept (section 5.7).

The chapter concludes with section 5.8, which provides an introduction into the testing case studies of chapters 6 to 8.

5.2 Requirements for domain specific extensions

We observed in the case studies of chapter 3 all the expected benefits and this demonstrated the usefulness of domain specific extensions. We even succeeded during the simulation studies in overcoming most of the risks of using domain specific extensions during the simulation studies. The risks that were still encountered during the simulation studies in chapter 3 were caused by a domain specific extension that was not fully suited for modeling the system, and not by properties of the domain specific extensions themselves. We believe that the risks could have been avoided completely during the

simulation study, if the domain specific extensions had been of even better quality. Quality improvement for a domain specific extension can be realized if the developers make sure that their domain specific extension matches the requirements listed in Table 5.1, which is based on the risks described in Table 3.7. These were based on the positive and negative observations of using domain specific extensions in the AGV case and the Airports case. The relationships between the requirements and the positive experiences gained during the case studies of chapter 3 are explained below.

Table 5.1: Potential risks of using domain specific extensions in a simulation study and requirements to mitigate these risks

Potential risk of using a domain specific extension in a simulation study	Requirements for a domain specific extension (DSE)
Activity 1. Problem description & define conceptual model	
Risk 1.1: scope of model developer is limited by model constructs	**Requirement 6:** Model constructs should be understandable for model developers
Activity 2. Select model constructs	
Risk 2.1: lack of trust results in no motivation to use domain specific extension	**Requirement 1:** DSE should show added value for model developers compared to use of model constructs of generic simulation environments
Risk 2.2: lack of insight in model constructs results in ignore domain specific extension	**Requirement 6:** Model constructs should be understandable for model developers
Risk 2.3: use of model constructs that are not suited for representation of system elements	**Requirement 2:** Use of model constructs of DSE should be clear and well defined so model developers know when and how to use the model constructs
Risk 2.4: system elements cannot be represented by model constructs	**Requirement 3:** System elements that seem to be exceptional for the domain represented by the DSE should not become model constructs
Risk 2.5: compose model constructs from developed domain specific model constructs only applied for infrastructure system elements	**Requirement 4:** The infrastructure and physical elements should be represented by model constructs separated from the model constructs for control or management
Risk 2.6: model developers can adjust internal logic of model constructs	**Requirement 5:** Internal logic of model constructs of DSE should be closed or accessible depending on type of model developer

(continued at next page)

Activity 3. Data collection	
No risks identified	
Activity 4. Instantiate simulation model for problem system	
Risk 4.1: model developers do not understand model construct	**Requirement 6:** Model constructs should be understandable for model developers
Risk 4.2: model developers do not know how to parameterize model construct	
Risk 4.3: difficult to compose simulation model, because model constructs are not available	**Requirement 7:** DSE should be an extendible set of model constructs
Risk 4.4: difficult to compose simulation model by person other than developer(s) domain specific extension	**Requirement 6:** Model constructs should be understandable for model developers
Activity 5. Verify and validate simulation model for original system	
Risk 5.1: mistakes of model developer are hard to overcome	**Requirement 8:** Behavior of model construct should be understandable and verifiable
Risk 5.2: model developers know something is wrong, but cannot identify what to do about it	
Activity 6. Analyze output of simulation model	
Risk 6.1: model constructs do not provide performance indicators problem owner desired	**Requirement 7:** DSE should be an extendible set of model constructs
Activity 7. Define solution for analyzed output	
Risk 7.1: model developers are triggered to find new solutions by parameters	**Requirement 2:** Use of model constructs of DSE should be clear and well defined so model developers know when and how to use the model constructs
Risk 7.2: model developers are limited by parameters and model constructs	**Requirement 7:** DSE should be an extendible set of model constructs
Activity 8. Instantiate simulation model for identified solution	
Risk 8.1: solution is identified that cannot be represented by model constructs	**Requirement 7:** DSE should be an extendible set of model constructs
Risk 8.2: adjustments of model constructs required to represent solution are time consuming	**Requirement 9:** Model constructs should be individually parameterizable
Risk 8.3: replacement of model constructs causes errors in model constructs that were linked or connected	**Requirement 9:** Model constructs should be individually parameterizable

Satisfying **requirement 1** helps model developers and problem owners to predict the benefits of using domain specific model constructs. Understanding the potential advantages will probably reduce reluctance to use a domain specific extension. The benefits of using a domain specific extension can be determined from experience gained in earlier studies using the same domain specific extension.

Requirement 2 implies that model developers who understand the capabilities and limitations of model constructs will be better able to select the correct model construct to represent an element of a system. The problem of selecting the correct model construct was satisfied by custom training during the simulation studies described in chapter 3. Additional support was provided at points where the model developers 'got stuck'. Improving the support and training should avoid this risk without the personal interventions of chapter 3.

We observed in the case studies for the airport, that the number of model constructs increased for every new part of an airport to be modeled. The consequence was that a lot of model constructs were part of the domain specific extension, but had only one instance in the simulation studies. The action of extending the set of model constructs was performed to overcome the risk that a system element was not represented as a model construct. The enthusiasm of the developer of the domain specific extension led in the case studies to large sets of model constructs after only a small number of simulation studies. Therefore **requirement 3** states that not every missing model construct should directly result in a new model construct in the domain specific extension. The aim of this requirement is to improve maintainability and use of the domain specific extension in subsequent simulation studies.

The AGV infrastructure in the simulation study described in chapter 3 was modeled using different hierarchical layers. Model constructs for the control of the AGVs and the allocation of AGVs and Loads to infrastructure were separated. The attempted separation of the infrastructure from the control and management enabled experimentation for just the infrastructure or just the control. Similar types of experiments can be identified in any system: either adjust the number of resources, infrastructure or physical elements in the system, or adjust the way the resources, infrastructure or physical elements are controlled and managed. **Requirement 4** implies that separation in classes of model constructs helps to achieve the flexibility required for experimentation.

The model developers who used the domain specific extension for airports were confronted with error messages that parameters were wrong. These model developers decided to adjust the code of the model constructs instead of adjusting the parameters in the simulation model. The effect was that the model constructs provided incorrect behavior and thus were no longer valid

system representations. **Requirement 5** indicates that model developers should be protected against such events.

We observed in the case studies that the development of a simulation model by the developers of the model construct went much faster and easier than model development by other persons. The model developers who were not involved in the development of the domain specific model constructs had start-up problems. The difficulties that model developers encountered during model development need to be reduced in such a way that model developers can carry out their work independently. Being able to understand the model constructs, as mentioned in **requirement 6,** is an important first step.

In the explanation for requirement 3 we state that not every system element should be directly transferred into a new model construct. However, it is also impossible to know the complete composition and size of the required set of model constructs in advance. Change is a given and therefore new model constructs will have to be added to the domain specific extension. Adding new model constructs should be possible without damaging the capabilities of the current model constructs and should not invalidate simulation models that have already been developed using the domain specific extension. The extendibility mentioned in **requirement 7** of the set of model construct is thus the ability to add or adjust model constructs of a domain specific extension.

One of the main advantages of domain specific extensions is that a model developer does not need to verify and validate the behavior of the model construct during the verification phase. He or she can safely assume that the model construct is working as defined and described. The laboratory experiments showed that the expert simulation model developers first blamed the set of model constructs and then secondly, started searching for mistakes in their own simulation model that used the model constructs. The possibility of verifying that the model constructs are technically correct will reduce this problem, and give us **requirement 8**.

Model constructs represent a part of a system. These system parts are different in each system, and also for the same system in different situations. **Requirement 9** enables the model developer to configure the model constructs in the simulation model to create an accurate representation of the system, without the model developer needing to know technical details about the inner structure of the model construct or the simulation environment.

5.3 Types of changes and extensions for domain specific extensions

The requirements for domain specific extensions can be satisfied in different ways. For example, requirement 6 “Model constructs should be understandable for model developers” can be achieved by training the model developer, by simplifying the model constructs, or by automating the process

of simulation model development so the model developer requires less understanding to use the model constructs. In addition each of these three solutions can be achieved in different ways and using different instruments. For example, the training of the model developer could be performed via a user manual, via assignments, using example models or by providing training videos for a specific model construct. Simplifying the model construct can be achieved via user interfacing, structural improvements, or better use of terminology. The automation of the process could be done by selecting the model construct out of a set, by automatic instantiation of the model construct in the simulation model or by automatic parameterization using information from a specific data source.

The example of how to tackle requirement 6 shows that there is not a single approach to satisfy the requirements. The case studies showed the effectiveness of activities to mitigate risks. We believe that the concept and guidelines described in chapter 2 can be improved to accommodate the actions required to mitigate risks before the risks occur. We set out, in Table 5.2, a list of changes and extensions to the concept of domain specific extensions that can be used to avoid the potential risks to be found during simulation studies.

Table 5.2: Changes to concepts and design guidelines of domain specific extensions to avoid risks

Requirement for domain specific extensions	Changes to concepts and guidelines for domain specific extensions as described in chapter 2
Requirement 1: DSE should show added value for model developers compared to use of model constructs of generic simulation environments	Additional documentation and example models; clear overview of capabilities and limitations of domain specific extensions
Requirement 2: Use of model constructs of DSE should be clear and well defined so model developers know when and how to use the model constructs	Additional documentation and example models; use terminology of domain
Requirement 3: System elements that seem to be exceptional for the domain represented by the DSE should not become model constructs	Structure of model constructs to enable exchange with generic model constructs of simulation environment; improvement to process to identify required model constructs in domain specific extension
Requirement 4: The infrastructure and physical elements should be represented by model constructs separated from the model constructs for control or management	Structure of model constructs to enable composition of predefined parts of model constructs.

(continued at next page)

Requirement for domain specific extensions	Changes to concepts and guidelines for domain specific extensions as described in chapter 2
Requirement 5: Internal logic of model constructs of DSE should be closed or accessible depending on type of model developer	Interfacing for model constructs to stop model developers of diving into internal logic
Requirement 6: Model constructs should be understandable for model developers	Training of model construct applicability; simplification of structure of model constructs; automate selection and parameterization of model constructs in simulation model
Requirement 7: DSE should be an extendible set of model constructs	Structure of model constructs to enable composition of predefined parts of model constructs; Structure of model constructs to enable exchange and/or co-operation with generic model constructs of simulation environment; Structure of model constructs to enable composition of predefined parts of model constructs
Requirement 8: Behavior of model construct should be understandable and verifiable	Example models including model construct; simple process description of using model construct together with other model constructs; documented specifications how model construct should perform under certain conditions; animation representing state of model construct in simulation model
Requirement 9: Model constructs should be individually parameterizable	User interface to model constructs

5.4 Simulation building blocks

5.4.1 Definition of simulation building blocks

Thus far we defined model constructs only as parts of a simulation environment. The domain specific model constructs that we introduced and used in chapters 2, 3 and 4, were developed using the ability provided by generic simulation environments to group generic model constructs. The result was a new model construct that could be instantiated in a simulation model to represent a system element. An analysis of the observations taught us that the concept of domain specific model constructs can make more

contributions to the effectiveness of simulation studies than the grouped generic model constructs can offer.

In this section we introduce a different structure and architecture for domain specific model constructs. The improvements in structure will allow changes in the model constructs to be made. The model constructs of domain specific extensions that follow this new structure and architecture will have the same advantages in simulation studies as we discovered in chapters 3 and 4. Model developers will be able to develop their simulation models faster, they will need less detailed programming and can more easily prepare their simulation model for new experiments. We will further refer to **model constructs** as the constructs of simulation environments that have been designed according to the principles of chapter 2. The constructs of domain specific extensions that follow the **new** structure and architecture will be called **simulation building blocks**. Simulation building blocks are extensions of model constructs of generic simulation environments, similar to the domain specific model constructs, but they have a structure that is expected to enable their use in future simulation studies in a better way and mitigate the risks encountered in the case studies of chapter 3. As simulation building blocks are extensions of generic model constructs, the simulation building blocks fit at exactly the same spot in Figure 2.2 at the position of model constructs P, Q, R & S.

Simulation building blocks are a product of the research of the BETADE-research program, a 5-year research program at Delft University of Technology that aimed at defining, specifying and using building blocks in different modeling domains, including geo-information, web-services and discrete event simulation (Verbraeck, 2002). The BETADE-research group provided the following definition for a building block: "*A building block is a self-contained, interoperable, reusable and replaceable unit, encapsulating its internal structure and providing useful services or functionality to its environment through precisely defined interfaces.*" (Verbraeck et al, 2002, p23).

The term **self-contained** in the above definition of a building block refers to the use of local information and local processes. Information within a building block represents the state of the building block and affects its behavior with respect to external events. This information is used for the processes and functions the building block performs. Once a building block receives an external event to execute a function, it can do this using the information and process descriptions that are part of the building block. For simulation building blocks this means that the building block keeps track of its own attributes and has all the knowledge and capabilities required to express the behavior of the system element it represents. We enable this by storing data locally in the simulation building block and dividing the functionality and services within a simulation building block over different elements, each with their own part of the system element's representation. These elements of

simulation building blocks will be referred to as **building block elements** and they provide the internal *structure* of the building blocks.

Interoperable means that the building block has to cooperate with other building blocks. This might seem to be contradictory to self-contained, but a simulation model cannot consist of just one super building block. System elements are represented by different simulation building blocks. These system elements together form the system to be simulated. These system elements in reality exchange information and entities, and the same has to occur between simulation building blocks. However, due to the self-containment of simulation building blocks, the way simulation building blocks are interoperable needs to follow certain rules, i.e. by precisely defined interfaces. The ability to be interoperable starts from the idea that simulation building blocks are part of a set. This set consists of a family of building blocks which are composed of the same type of building block elements, e.g. the 'area' in the airport case study of chapter 3. The set of simulation building blocks consists of building blocks representing infrastructure and building blocks for control or management. The simulation building blocks for control are represented as a process description and use pointers to infrastructure building blocks, in order to be interoperable with simulation building blocks for infrastructure or physical elements.

Reusability for a simulation building block means that the simulation building blocks are instantiated more than once in a simulation model, or the simulation building block is instantiated in simulation models used in several simulation studies. When a simulation building block is reused in a simulation model several times, it is especially necessary to be able to parameterize the simulation building block according to the properties and behavior of the system elements it represents. Internal reuse is also achieved by the use of building block elements within a simulation building block, which enables flexibility and the ability to extend the set of building blocks with new building block elements instead of directly developing new building blocks.

A building block is **replaceable** if it can be removed from a system and another building block can take its place in the system. The system should still work after the change. For example, replacing a CPU in a computer by a newer model will give the result that the computer is still capable of executing the same software, but it can do it faster. The same applies for simulation building blocks. Replacing one simulation building block by another building block is a type of experimenting to evaluate system alternatives. The replaceability is achieved via the family of simulation building blocks that operate together with other simulation building blocks via standardized interfaces. Of course the replaceability is only possible between carefully designed sets of building blocks that have the same interface and a similar function. When looking at the CPU example: only a CPU of the same family and with the same pin structure can potentially replace the original CPU.

Encapsulating its internal structure means that the model developer does not need to know what is inside a simulation building block. The internal working of a simulation building block is shown to the outside world by expressing the state of the simulation building block. The user interface will be the only thing the model developer will observe, but this can vary depending on the type of model developer, e.g. allow expert users a peek inside.

Building blocks are part of a system for a number of reasons: these are the **useful services or functionality** that the building block provides to a system. Each simulation building block in a simulation model should add something to the overall system representation; otherwise it can be left out. In addition, the services a simulation building block provides is to other building blocks in the system. A separation can be made between the type of services and functionality and how they are allocated to simulation building blocks, because no simulation building block in a domain specific extension provides all the services and functionalities of the system. Several services or functionalities will be offered by building block elements inside the simulation building block. In some cases the desired service or functionality can only be realized by the model developer by integrating the building block with model constructs of the generic simulation environment

Building blocks encapsulate their internal structure and are self-contained, yet they provide services to other building blocks and are interoperable. This means that somehow they exchange information. Therefore, building blocks have **precisely defined interfaces**. A building block contains several types of precisely defined interfaces for different purposes. These purposes are exchange of information and entities with other building blocks, parameterization by the model developer and collection of statistics at the end of the simulation run. A part of the statistics and external representation of the simulation building block is also realized via an interface for visualization and animation.

The BETADE research group described the use of building blocks in the following way: "*A building block may be customized in order to match the specific requirements of the environment in which it is 'plugged' or used.*" (Verbraeck et al, 2002, p23). This means, for the use of domain specific extensions, that building blocks are instantiated into a simulation model and within the simulation model the simulation building blocks will be parameterized to represent a system element.

The BETADE research group defined building block as widely applicable, beyond the use of a single domain like software engineering, data management or simulation. Barros et al (2004) summarize the difference between component based software development and component based simulation. They conclude that a simulation component or building block behaves differently than a software component, due to level of abstraction and modeled system representation. Nevertheless, they conclude that concepts

like separation of concerns, interfacing and product line engineering can also be applicable for the development of components for simulation.

5.4.2 The way that simulation building blocks achieve 'self-contained'

Data locally stored

The characteristic of a building block that it is self-contained relates to information belonging to the simulation building block. The state of a simulation building block is defined as the values of all its attributes (sometimes called state variables). Examples of these attributes are the current destination or the current speed of an AGV. The attributes of a simulation building block will vary over time. The AGV will change destination and speed regularly. In most generic simulation environments, the speed attribute of a vehicle can be changed by *any* model construct. Freely changing the attributes of other building blocks often results in unexplainable behavior and difficulties with verifying and validating a simulation model. Therefore it is important that a simulation building block only changes its *own* attributes. For example, a control simulation building block should send a message to an AGV if the control simulation building block has decided that the destination of the AGV needs to change, instead of directly overwriting the destination in the AGV building block.

Simulation Building Block Guideline 1: data belonging to a building block should not be written by other building blocks directly, but only via defined interfaces.

Use of building block elements

The 'self-contained' characteristic does not just deal with the need for shielded attributes of the simulation building blocks to resemble their state, it also applies to the functions or processes within building blocks. For example, the area model construct in the airport case study (chapter 3) had at least the following functions: resource limitation to entering passengers; prioritization of passengers queuing for the area; determination of duration passengers remain in area; collecting of statistics of passenger in area; providing a relation to other areas for using shortest path algorithm. 'At least' is used because the advanced areas contained mechanisms to e.g., represent conveyor belts, to adjust the capacity of resources in time and the ability to trigger changes of attributes for passengers.

Each combination of functionalities results in a new model construct to represent an area according to the specific requirements of the system. An example of an area with several functionalities is the check-in area, i.e. the area where passengers hand over their luggage and receive a boarding pass. The functions of this area are a priority mechanism for queuing passengers, a calculation for the duration of the processes depending on the attributes of a

passenger, a mechanism to change attributes of passengers at the counter, and a mechanism to change operator availability depending on queue size or allocated flights. At different airports the check-in areas were slightly different and thus different model constructs were developed of the check-in areas resulting in a large set of variants.

The issue with the variants used to model constructs has also been encountered in component development in software engineering. Software component developers overcome the large sets of variants by applying the concept of Product Line Engineering (Weis and Lai, 1999). In this concept the functionalities of components are divided over smaller objects. One of these objects contains the core of the component that will be the same for all variant components. The other objects are adjusted to represent the different variants. Figure 5.1 shows, on the left side, three software components, before Product Line Engineering is applied. In these three components you can observe three objects (triangle, circle and diamond). The circle object is the core of the component, the triangle and diamond objects are variants for respectively function X and function Y. Product Line Engineering enables us to find the commonality in the component variants and combine them. Figure 5.1 shows, at the right side, the same three software components, but now the engineering concept is applied and maintenance and usability is focused on the core of the component and its smaller objects that provide the variations.

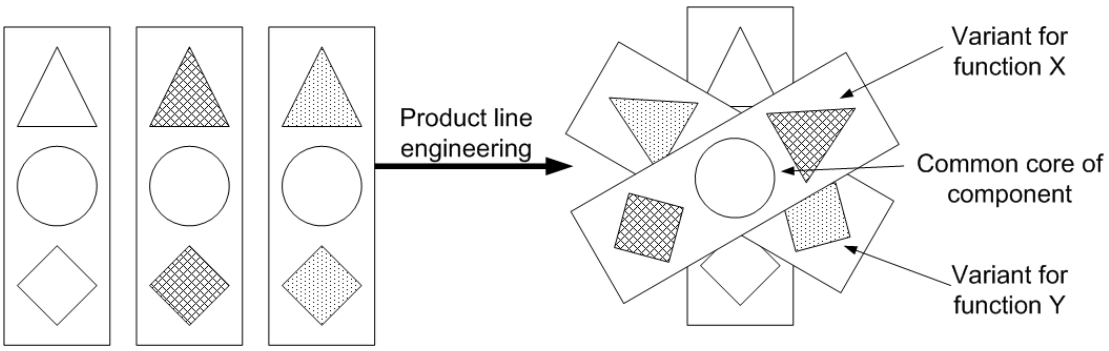


Figure 5.1: Product Line Engineering; find commonality in alternative components

The concept of Product Line Engineering can be applied for each simulation building block, where one or more functionalities can be observed as a functional variant. Product Line Engineering is worth applying if variants can be implemented by smaller elements to represent a slightly different behavior. The check-in counter model construct is a good example of such a future simulation building block as it has specific functionalities for the queuing mechanism, processing, resource availability and passenger state changes, where other area building blocks have different implementations for these same functionalities. We model these functional variants as specific objects,

which will be a part of the simulation building blocks. We will refer to these specific objects as *building block elements*.

The simulation building block will consist of one of the instances of building block elements for each functionality in this example. The building block elements might vary for building blocks in functionality and some might not even be available. In the example of Figure 5.2 the simulation building block with the circle as core will be instantiated with one of the triangle and one of the diamond building block elements. The model developer can thus make a decision for one of the available building block elements for each functionality.

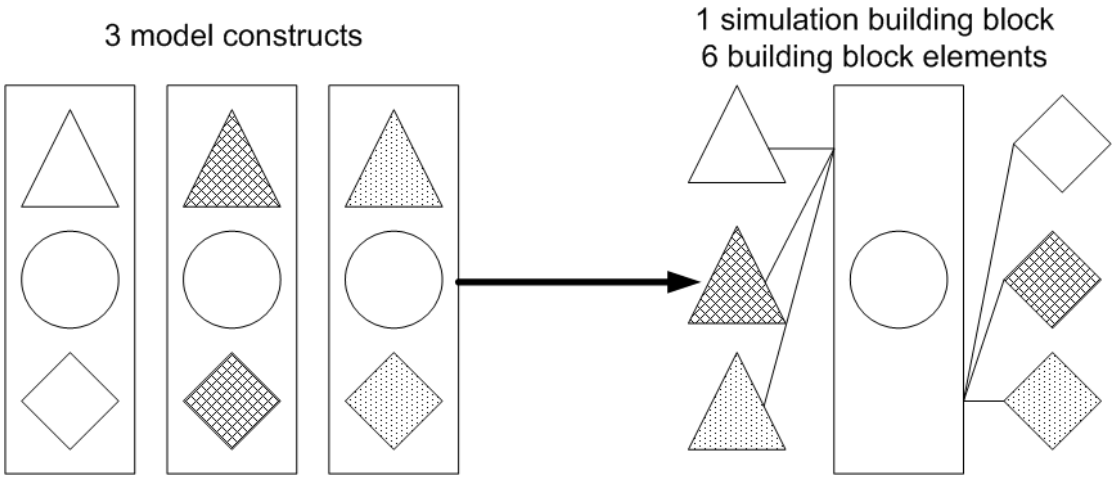


Figure 5.2: 3 model constructs versus 1 Simulation building block with a circle core and 6 building block elements to represent system elements.

The use of building block elements enlarges the number of system elements that can be represented by the domain specific extension. Figure 5.2 shows that originally only 3 model constructs were available, and thus only 3 representations of the system elements were available. With 6 building block elements the model developer can now represent 9 (3 triangle * 3 diamond) system elements.

Simulation Building Block Guideline 2: a simulation building block consists of a core and building block elements to represent functions and services.

The building block elements that are part of a simulation building block will vary with each domain specific extension. However, we encountered in all simulation studies we were involved in the need for resourcing behaviour, the requirement for statistics and the need to handle errors in the execution. Based on the experience we gained in simulation studies we envision that the building block elements listed in Table 5.3 are likely to return in many simulation building blocks. These common building block elements provide extra richness to the concept of simulation building blocks compared to model

constructs and are identified to help the design of the simulation building block and its building block elements.

Table 5.3: Building block elements expected in simulation building blocks

Building block element	Description
Statistics	Building block element to calculate statistics like resource utilization, waiting time and process duration of functions and services provided by the simulation building block.
Error handling	Building block element to handle errors based on wrong or inconsistent input, or based on a wrong or inconsistent state.
Resource definition	Definition of the availability and capacity of the resource represented by the simulation building block.
Evaluate capacity availability	Building block element to check whether a resource of the building block currently has enough free capacity.
Claim capacity	Building block element to claim capacity of the resource of the simulation building block. This building block element also can include queueing and/or storing functionalities.
Carry out process	Building block element to trigger the simulation building block to perform an allocated function or service. The process can be executed for a certain (random) duration or until a certain state is reached.
Release capacity	Building block element to release capacity that earlier has been claimed by a process to ensure execution of service or function.
Animation	Building block element to animate the state of the simulation building block, possibly including one or more of the statistics of the simulation building block.

The building block elements that are described in Table 5.3 can use a variety of attributes of the simulation building blocks to perform each function or service. The attributes used, and the information to be exchanged in these building block elements, are available within the simulation building block. A developer of simulation building blocks might decide to introduce a reduction of data exchange of building block elements as described in Simulation Building Block Guideline 1, i.e. data exchange requires the use of defined interfaces. However, this will strongly reduce the understandability of the inner logic of the building block element and is therefore seen as an unnecessary overhead.

Simulation Building Block Guideline 3: data belonging to a building block element can be accessed by other building blocks elements of that building block without using the interfaces of the simulation building block.

5.4.3 The way that simulation building blocks achieve 'interoperable'

Set of simulation building blocks

The IEEE (1990) defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged." The components that the IEEE talks about are software components instantiated in a software application, but the definition also applies if we replace software component by simulation building block or building block element. The ability of simulation building blocks to exchange information only applies for simulation building blocks that belong together, i.e. simulation building block members of a domain specific extension. The set of simulation building blocks in a domain specific extension can be structured according to different views.

One view is the identification of families of simulation building block. We define a **family** as a set of simulation building blocks that all represent a type of system element. The areas in the airport case study are examples of simulation building blocks that can be structured as a family. The use of families is to support the simulation model developer in understanding the replaceability of a simulation building block by another simulation building block from the same family.

Another view on the structure of simulation building blocks is the difference between infrastructure or physical building blocks and control or management building blocks. The infrastructure and physical building blocks represent the system elements that deal with the processing and handling of entities, while the control and management building blocks deal with the allocation of entities to infrastructure or physical building blocks or triggers when physical building blocks should start or stop processes.

The third view of the structure is the view of fixed control or control using process steps. Within a system a distinction can be made between control or management systems that can be modeled as process steps in a fixed order, or by a much more flexible solution where the process can be defined stepwise. For example, the allocation of AGVs to a dock was, in the case study in chapter 3, a fixed sequence, while the actual claiming of tracks to drive on was defined in a process sequence using scripts.

Family of building blocks with building block elements

The areas in the domain specific extension for airports are a clear example of a family of simulation building blocks. The simulation model was configured by instantiation of various members of the area family, for example the walking area, the conveyor, the shop, the boarding area and the check-in counter. Possibly several other families can also form part of the simulation environment. For example, a second family in the domain specific extension for airports could be the building blocks implementing mechanisms of allocating airlines to check-in counters, a set of building blocks that was developed for the simulation study at JFK Terminal 4.

The simulation building blocks that belong to a family will have mostly the same types of building block elements. Figure 5.3 shows 3 simulation building blocks based on the simulation building block introduced in Figure 5.2. Exactly as shown in Figure 5.2, the building blocks of that family share one or more types of building block elements and further have variants to a basic simple version of building block elements. Figure 5.3 shows a part of an example domain specific extension consisting of one family of simulation building blocks and building block elements based on the Product Line Engineering example of Figure 5.2. Families are to structure the available simulation building blocks to support the model developer. Families can be organized in any way that the model developer and the future users feel comfortable with.

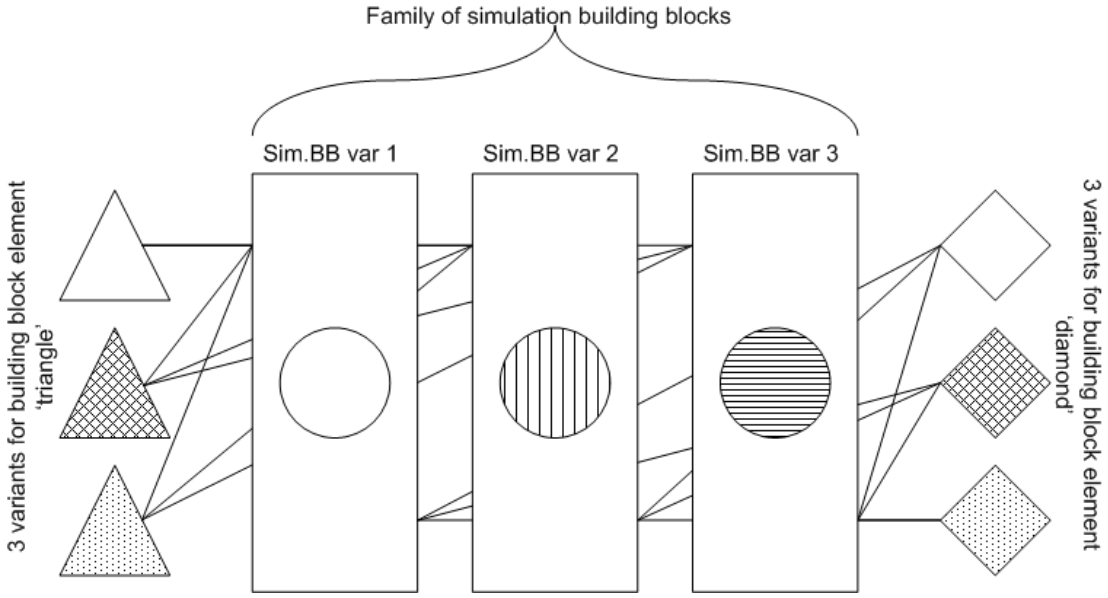


Figure 5.3: Family of simulation building block extended of figure 5.2

Simulation Building Block Guideline 4: system elements that appear in different variants and processes in a system can be organized in families of building blocks and building block elements.

Building blocks for infrastructure and building blocks for control

Two types of experiments are often performed in simulation studies to improve system performance. The first option is to extend the availability of resources, the second option is to improve the way that resources are used: by resources we mean things such as machines, vehicles or people. We refer to these items as infrastructure or physical elements of the systems. The infrastructure or physical elements carry out processes for other elements in the system. For example, the AGVs perform a process for a Load or an Area performs a process for a Passenger.

The processes, services and functions performed by physical elements or infrastructure are determined by control or management functions in the

system. For example, a management system allocates which Load an AGV will transport. Simultaneously a control system makes sure that the AGV can safely move over a Track. The control system triggers when the AGV can start driving, and the AGV notifies the control mechanism when it reaches the end of the provided Track.

Figure 5.4 shows the structure of control and infrastructure building blocks in generic terms. The infrastructure simulation building blocks will be triggered and provide feedback to the control simulation building block. The introduction of these two types of simulation building blocks, control and infrastructure, support the model developer in performing simulation experiments to vary the availability of infrastructure and to adjust the control of the infrastructure, see also Saanen (2004) and Versteegt (2004) who further describe the use of control and infrastructure dedicated building blocks.

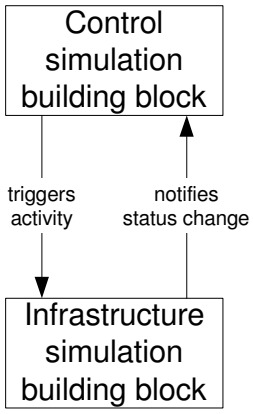


Figure 5.4: Simulation building blocks that separate the control and infrastructure

Simulation Building Block Guideline 5: building blocks are of different types, it is common to have separate building blocks for infrastructure and for control.

Process description for control simulation building blocks

The way the infrastructure is controlled and managed is in each system different. Therefore simulation building blocks need to have a way of flexible control by other simulation building blocks. A part of the required flexibility in control will be achieved via building block elements within control building blocks. The scripts in the AGV case study provided an alternative way for controlling the infrastructure. The control consisted of a sequence of processes instead of control that consisted of one model construct. This approach resulted in more flexibility for the control of AGVs, and the ability to model control matching to every possible layout of tracks.

In the AGV case study flexibility was achieved by a scripting language that was hard to maintain and limited in configuration. The different script statements can be seen as individual simulation building blocks consisting of

the building block elements like interaction, statistics, error handling and one or more building block elements for the actual process and control to the infrastructure. Table 5.4 shows a fragment of an actual script and Figure 5.5 shows how this script could be instantiated in a simulation model using different control simulation building blocks.

Script LR	Comments
Insist SX	Claim ticket for crossing
Exec AX	Drive from left to center
Exec XB	Drive from center to right
Free SX	Free ticket SX

Table 5.4: Script of AGVs in table

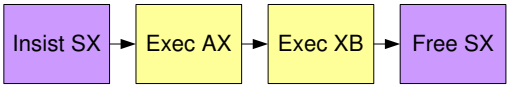


Figure 5.5: Script of AGVs using simulation building blocks

Simulation Building Block Guideline 6: complex control mechanisms should be represented using control building blocks linked together to represent a flow.

Pointers between simulation building blocks

Interoperable means that the building blocks work together and exchange information. The building blocks that are instantiated in the simulation model can only interact if they are aware of each other's existence. The simulation environments offer several technical ways to achieve such awareness, i.e. to know that other building blocks exist in the simulation model and what their names or identifiers are. The easiest way is to point to the other building blocks via their name. This process was applied to the simulation models within the AGV case study and resulted in several errors so that model constructs were not available or did not receive the correct name. We learned that the awareness of other model constructs should be flexible and not fixed in the design of the model construct.

We introduced flexibility in defining the pointer to other building blocks in the airport case studies. For example, in the model construct to allocate flights to a check-in area we listed the check-in areas using a pointer. These parameters were verified and error messages were generated if the pointer was invalid. In this way awareness of other model constructs was easier to obtain, check and update.

Awareness of the simulation building blocks can be achieved via pointers, but it is very time consuming to set the pointers of each building block to point to other building blocks it might interact with. An alternative approach is to use one instance of a specific building block that contains pointers to all building blocks in the model. This mechanism can be compared to a naming function in software engineering.

Simulation Building Block Guideline 7: building blocks should be aware of each other's existence within a range of applicability.

5.4.4 The way that simulation building blocks achieve 'reusable unit'

The reuse of a simulation building block will be improved by the use of building block elements. A simulation building block that exactly fits system A does not need to be a good match for system B, even though the domain is the same. The reuse of simulation building blocks is less of an issue, because the building block that was a good match in system A can be updated to match system B via adjustment, replacement or alternative parameterization of one of the building block elements in the simulation building block, see also Simulation Building Block Guideline 2.

5.4.5 The way that simulation building blocks achieve 'replaceable unit'

Family of building blocks

Replacing a simulation building block in a simulation model can be easily achieved for building blocks that are part of the same set. A check-in counter in the simulation study of the airports could easily be replaced by a check-in counter with another function or the waiting area could be replaced by a shopping area to entertain passengers during waiting.

Replacing of elements in the simulation model can be achieved by using simulation building blocks from the same family or by replacing building block elements in the instantiated simulation building blocks. This works exactly as defined in Simulation Building Block Guideline 4.

Extend set with new building block elements

The use of building block elements also improves our ability to extend the set of building blocks. In cases where a model developer cannot find a matching building block to represent alternative behavior in the system, then a new building block element can resolve the problem. The domain specific extension will include the new building block element and the model developer can then develop simulation models with new simulation building blocks that were not considered in the original design of the domain specific extension.

Simulation Building Block Guideline 8: extension of a domain specific extension can be achieved by introducing new building block elements for existing simulation building blocks.

Standardized interfaces

Replacing building blocks in a simulation model is not a direct result of using families of building blocks, it is due to the building blocks having the same interface within the family. By interface we mean the way building blocks interact with other building blocks. If a control building block expects a check-in area to send triggers and receive notifications in a certain way, then it is important that a building block that can replace the original check-in area handles the same type of interaction via the same interfaces. This interface is the way that building blocks receive triggers to do things. Standardized

interfaces to enable replaceablity applies for building block elements in exactly the same manner as for simulation building blocks, because building block elements should also be replaced without any complexity for the model developer.

Simulation Building Block Guideline 9: simulation building blocks and building block elements of the same family follow the same interface requirements.

5.4.6 The way that simulation building blocks achieve 'encapsulating its internal structure'

Hide inner working

The internal structure of simulation building blocks and its underlying building block elements are hidden behind a user interface to the simulation building block in which the model developer enters the values for parameters. This user interface completely hides the inner code and prevents the model developer seeing what is inside or even tampering with the logic of the simulation building block or building block elements inside.

Simulation Building Block Guideline 10: simulation building blocks hide their inner working.

Limitations depending on type of model developer

We observed during the laboratory experiments that experts want to understand the inner workings of the set of model constructs and see what is hidden behind the interface. We also observed that the developers of domain specific extension made improvements in the model constructs instantiated in the simulation in the case study of the AGVs. We conclude from these two observations that the user interface is a good way to hide the inner working, but that, depending on the type of model developer, a way to observe the inner working of the building blocks might be desired. Especially in the development of the simulation building blocks and building block elements advanced developers will want to test and verify a block by evaluating the detailed logic of the simulation building block or the building block element.

Simulation Building Block Guideline 11: advanced model developers have to be able to unhide the inner logic and see how the processes and attributes are implemented.

5.4.7 The way that simulation building blocks achieve 'providing useful services or functionality'

Simulation building block elements

The services and functionalities that simulation building blocks provide are represented by building block elements. A simulation building block can have several building block elements for the different services or functionalities it provides. This follows from Simulation Building Block Guideline 2.

Integrate with model constructs of generic simulation environment

The service or functionality that should be represented by a simulation building block can be exceptional compared to the common representation of the system element in the domain. In systems were a system element needs to be represented that is exceptional for the type of systems in a domain, then it might not be worth extending the domain specific extension with new building block elements. An alternative approach to Simulation Building Block Guideline 8 is to instantiate the desired functionality of the building block via custom model constructs of the generic simulation environment.

The breakdown process of the computer factory consists of three building block elements: "Generate a breakdown", "Call an operator", "Solve the breakdown". Sufficient building block elements have been developed for the sub-process of "Call an operator" and "Solve the breakdown", but in one factory a machine is in use that generates breakdowns every 24 hours but only if more than 200 boxes have been closed in the past hour and only if preventive maintenance has taken place more than 3 hours ago. Figure 5.6 shows the simulation building block "Breakdown process" instantiated using 4 generic model constructs to represent the very specific building block element.

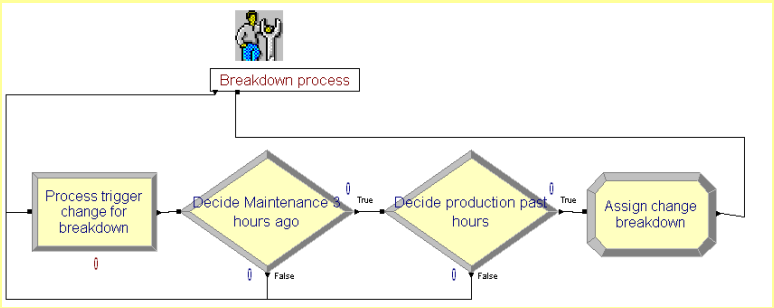


Figure 5.6: "Breakdown process" using several generic model constructs for the building block element "Generate breakdown"

Applying building block elements for functionalities reduces the chance to encounter the risk of developing building blocks to represent exceptional system elements, but still there is a risk that the developers of the simulation building blocks and building block elements will spend their time on a building block representing an exceptional system element that is not required in other simulation models. These exceptional system elements can much better be represented by one or more generic model constructs instead of a dedicated domain specific simulation building block. We suggest, therefore, that a model developer should have the possibility to use generic model constructs to represent a certain system element or part of a system element. The block

below and Figure 5.6 show an example of a simulation building block that requires one of the building block elements to be represented by generic model constructs. The discussion above results in Simulation Building Block Guideline 12.

Simulation Building Block Guideline 12: system elements should be represented by building block elements that can be extended with custom instantiations of model constructs of a generic simulation environment.

An alternative approach is to enable simulation building blocks to interact with model constructs of the generic simulation environment. This approach will be mainly applied to the control building blocks represented in a process description according to Simulation Building Block Guideline 6. Figure 5.7 provides an example of simulation building blocks from a domain specific extension (rectangle with thin border) with model constructs of the generic simulation environment (shapes with thick 3D-border). The ability to interact with the generic model constructs of the simulation environment gives the result that the domain specific extension really is an extension to the simulation environment and not a partial replacement.

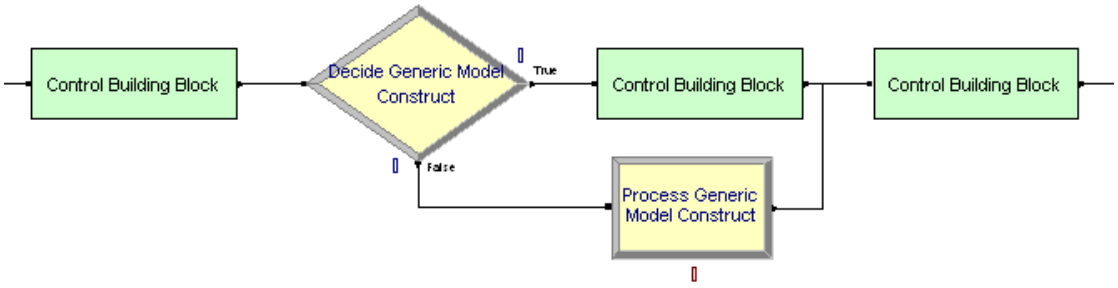


Figure 5.7: Simulation building blocks and generic model constructs integrated

The difference with Simulation Building Block Guideline 12 is that in Figure 5.7 the building blocks and generic model constructs jointly are composed into the model and that without the combination the simulation model will not work. In Simulation Building Block Guideline 12 the assumption is that the simulation building block can work without the generic model constructs, but that the generic model constructs represent a specific or exceptional variant that is not worth the development of a building block element as part of the domain specific extension.

Simulation Building Block Guideline 13: a building block can connect to model constructs of a generic simulation environment.

5.4.8 The way that simulation building blocks achieve 'precisely defined interfaces'

Different interfaces of simulation building blocks

A simulation building block contains interfaces that serve different aims, during model development, during the simulation run and during the analysis of the simulation experiments. The interface with the simulation model developer, mainly during model development, is used to support understanding of the possible use of the simulation building block and to enable the model developer to easily parameterize the simulation building block. The interfaces used during the simulation run are interfaces to other simulation building blocks, used to exchange information and to trigger functions and processes in the other building blocks in the simulation model. Finally, the interfaces for analysis are used by the analyst during the execution of the simulation run and after the simulation run has been completed. The interface is a visualization of a selected subset of the state of the simulation building block during the simulation run. The interface contains reporting of statistics after the simulation run is completed, gathered from the state of the simulation building block during the run.

We first describe the interface used by the model developer for parameterization, which is usually built using the features of the generic simulation environments to offer dialogs and hide the inner working of the building block, see also Simulation Building Block Guideline 10 and 11. Secondly, we describe how simulation building blocks should exchange information. This is mainly an approach to be followed by the developer of the domain specific extension, because most generic simulation environments do not force this on the developers. This also follows Simulation Building Block Guideline 1, i.e. that exchange between building blocks should go via their interfaces, and Simulation Building Block Guideline 9, i.e. that building blocks of the same family follow the same defined interfaces. Thirdly, we describe how the information on performance indicators can be visualized or reported to the model user.

User interface parameter settings

In chapter 2 we showed the differences between interfaces of generic model constructs (Figure 2.5) and the interface of a domain specific model construct (Figure 2.6). Simulation building blocks require a similar parameterization to represent a system element of a specific system, see Figure 5.8.

The user interface has to contain fields in which the model developer can enter e.g., the parameters or statistical distributions for processes, the availability of resources, or the initial state of a simulation building block. The values that the model developer enters in the user interface of the simulation building block will be used during the simulation run and for the initialization of the simulation building block. The model developer does not need to make

adjustments in one or more places of the underlying code thanks to the user interface, but can rely on the simulation building block to use the values entered in this user interface.

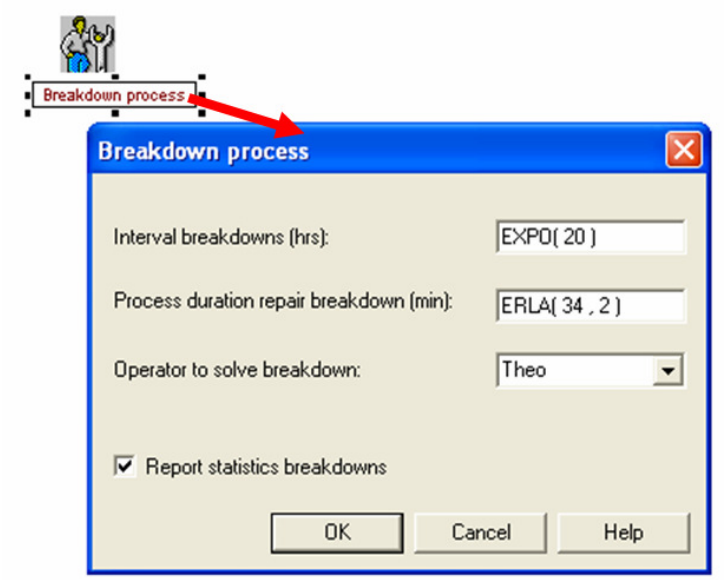


Figure 5.8: User interface for a simulation building block

Simulation Building Block Guideline 14: the model developer has to adjust the parameters of a simulation building block via a user interface.

The user interface has an additional advantage that the developers of the simulation building block should use. In a generic model construct the user interfaces are kept generic. For example, a process model construct uses the term “Process duration” while the process model construct refers to the process used to repair a breakdown. Figure 5.8 shows that the simulation building block can use the text “Process duration repair breakdown” to clarify what process parameter is entered. Terminology in the user interface clarifies the use and capabilities of the simulation building block within a domain and supports the model developer and helps him/her to make correct choices for configuring the building blocks in a simulation model.

Simulation Building Block Guideline 15: use of domain terminology in the user interface provides insight in the suitability of a building block for a certain purpose and the meaning of its parameters.

Model developers will use the user interface of the simulation building block to enter their data. When they are feeding their data into the simulation building block, a check can be performed automatically to verify the correctness of the parameters. The simulation building block can provide warnings or error messages to the model developer if a parameter of the

simulation building block is not within a valid range. For example, the interval breakdown should always have a positive value in the interface in Figure 5.8.

Simulation Building Block Guideline 16: parameters in a user interface of a simulation building block have to be checked for validity of the values.

The parameters that a model developer uses to configure an instance of a simulation building block in a simulation model will differ between projects. In some projects with a domain specific extension all parameters will be custom defined, in another simulation study it is sufficient to work with average values from the domain. For example, the walking speed of passengers in an airport can be varied in each simulation study, but in some studies it could be sufficient to take an average value such as 4.5 km/hr. If the developer of the simulation building block has already put in a default value (for example 4.5 km/hr.) then the development of a simulation model is easier for a model developer.

Simulation Building Block Guideline 17: parameters in a user interface of a simulation building block should have default values whenever possible.

Documentation is an important instrument which model developers must study to understand how a simulation building block should be used. However, model developers will usually not spend hours reading a user manual. They need to have easy access to relevant support and explanations that link to the simulation building block on which they are working. The simulation building block in Figure 5.8 offers this easy access to support and documentation via a help button in the user interface. Pressing the help button should return information that explains, for the model developer, how to use the simulation building block. This information could be, for example, a page number from the user manual, a window, or a web page.

Simulation Building Block Guideline 18: The user interface of a simulation building block should provide support for the model developer.

The building block elements that represent functions and services of the simulation building block are hidden for the model developer within the simulation building block. Different technical solutions can be used to enable the model developer to change building block elements depending on the generic simulation environment used, but whatever technical option the generic simulation environment offers, the model developer will have to make these changes via the user interface. In the user interface of the simulation building block the user can select the appropriate building block elements and set their parameters correctly. The user interface is thus not only used for

parameters of the simulation building block, but also for changing parameters of the underlying building block elements.

Simulation Building Block Guideline 19: The user interface of a simulation building block can be used by model developers to select building block elements and set their parameters.

Interaction using designed interfaces

Handling interaction between building blocks is one of the main features required to successfully replace simulation building blocks in a simulation model. A building block in a simulation model cannot be replaced by another building block if the two building blocks use different ways of interacting with the rest of the building blocks in the simulation model. Figure 5.9 is a schematic representation of the interaction issue as it appeared in the AGV case study. On the left hand side the interaction between different model constructs of the Terminal Manager (star at bottom) is shown. In the right picture an alternative Terminal Manager model construct is used, which needs another type of interaction by one of the surrounding building blocks. The model construct can thus not be replaced without changing the interactions with other model constructs.

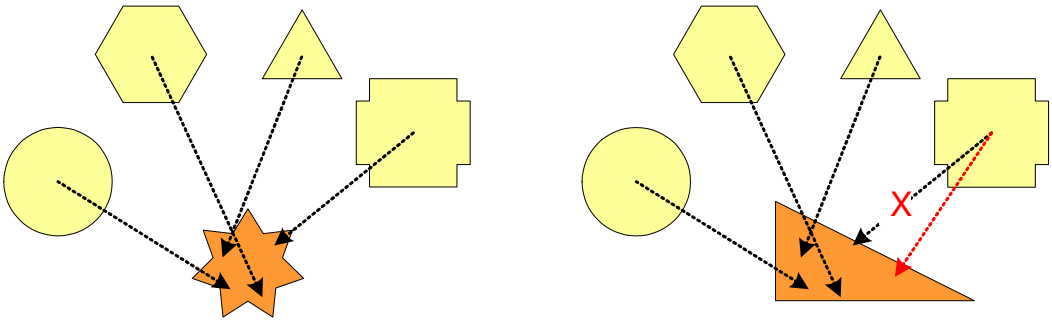


Figure 5.9: Unstructured interaction model constructs

In software development (Meyer, 1997; Szyperski, 2001; Atkinson et al, 2001) this issue with interaction has been tackled in two steps. The major step is to create awareness during the development of the software applications. Software developers force themselves to make sure that their components can follow a structure that has been agreed with all the developers. This way of developing software components is known as Design By Contract or, Programming By Contract (Jézéquel and Meyer, 1997; Atkinson et al, 2001). The second step is the use of development environments and languages that force the use of well-defined interfaces.

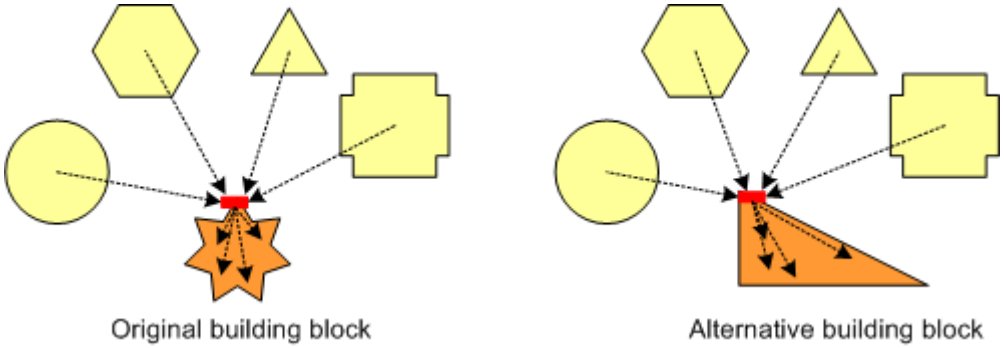


Figure 5.10: Structured interface for simulation building block

The effect of these interfaces is shown in Figure 5.10. Other simulation building blocks no longer interact with something inside the model construct, but send their trigger, request, entity, or event to the interface of the simulation building block. The simulation building block will redistribute the received object internally to the appropriate building block element. The original building block on the left hand side in Figure 5.10 can be replaced by an alternative building block (right hand side of Figure 5.10) without any changes to the other simulation building blocks in the simulation model.

In Figure 5.10 only one interface remains and all interaction is managed via the small bar above the star or triangle. The important thing is that all the building blocks in the family follow the same interface, it does not mean that this is the only point of entry for the building block. The interface can easily be distributed over two or three different locations, but the more points of entry are available, the larger the chance that developers of the simulation building block will ignore the defined interface and try to connect directly, therefore we suggest to work with one point of entry.

Simulation Building Block Guideline 20: a simulation building block has a defined interface that receives triggers, requests, entities, or events from other simulation building blocks in the simulation model and redistributes these internally.

The interfaces have an additional function besides redistributing. They can also be used to check the state of a simulation building block before carrying out the logic associated with that trigger. For example, an AGV cannot receive the trigger “pick up load” if it is currently in maintenance. Another check that the interface can perform is to capture triggers that are not supposed to be received at all by a simulation building block. For example, the model developer may have made some errors with the pointers result that a repair man has to pick up a load. The interface of the repair man is not prepared for the trigger “pick up load” and thus the event is cancelled and a runtime error message is provided to the model developer.

Simulation Building Block Guideline 21: the interface of a simulation building block contains evaluations of the state of the trigger and the building block to determine whether the building block can handle the trigger.

User interface visualization

The last type of interfacing of a simulation building block is the interfacing it provides to the analyst to allow them to evaluate the behavior of the model. This interface is used during the simulation run or afterwards to visualize the states and key performance indicators.

Visualization during the simulation run can be provided by animation elements that are part of the simulation building block. Figure 5.11 shows animation using a set of pictures. Depending on the state of the simulation building block a different picture is shown. Other options are counters and texts that show, for example, the number of entities sealed at the building block or the number of breakdowns that occur. The important thing is that all these animation elements are provided to the model developer at the moment that the simulation building block is instantiated. Once again, different simulation environments will implement this in different ways, but independent of the simulation environment, the simulation building block should already contain the visualization definitions.

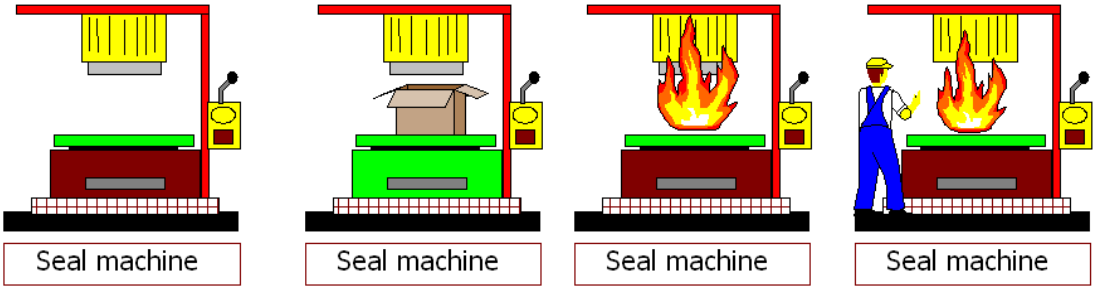


Figure 5.11: Four status visualizations of the seal machine: available, sealing a box, break down, repairing a breakdown

Simulation Building Block Guideline 22: a simulation building block contains pictures, texts, numbers and other elements to support visualization of the state, and the key performance indicators during the simulation run.

5.5 Additional tools for domain specific extensions

The use of simulation building blocks in simulation projects standardizes the way the simulation models represents a system in a certain domain. The simulation building blocks will also collect statistics in a standardized way. The standardization of the simulation building blocks in a domain specific extension enables the automation of the steps a model developer has to perform.

The complete process of performing a simulation study is shown in Figure 1.8. The following steps are part of this process and can be supported by automation and additional support tools on top of the set of simulation building blocks and the generic simulation environment:

- Model development by instantiating model constructs
- Set parameters of the model constructs according to system data
- Verify logic and data entry of simulation model before running
- Run simulation experiment
- Analyze output of the simulation model

5.5.1 Model development by instantiating model constructs

The simulation building blocks can be automatically instantiated by an application. The possible automatic development from, e.g., a Visio drawing by a tool that instantiates the building blocks at the correct location into the simulation model is shown in Figure 5.12.

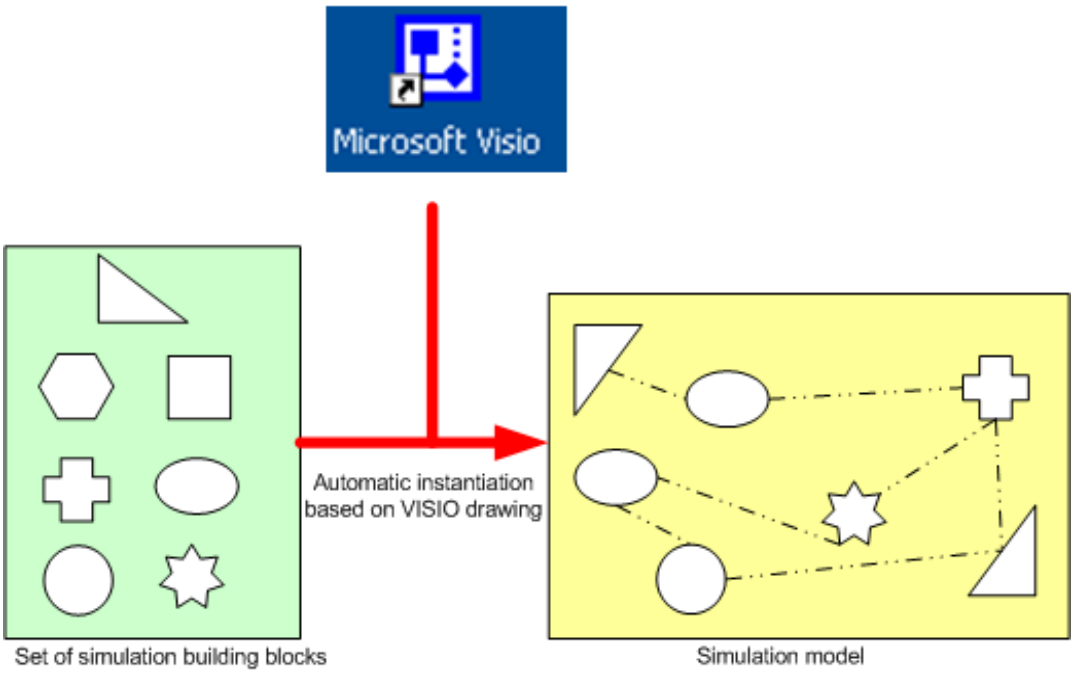


Figure 5.12: Automatic model instantiation

5.5.2 Set parameters of the model constructs according to system data

Instantiating simulation building blocks is a one-time activity in model development. The main effort is the parameterization of the building blocks to represent slightly different systems. Simulation Building Block Guideline 14 describes how parameters of a simulation building block can be set via a customized user interface. Standardization of the interface for multiple building blocks in a simulation model enables us to define a common user interface covering the complete simulation model.

This simulation model interface can be provided in different ways, for example a spreadsheet or a database application. An example spreadsheet interface that enables the model developer to set parameters of multiple simulation building blocks in the simulation model is shown in Figure 5.13.

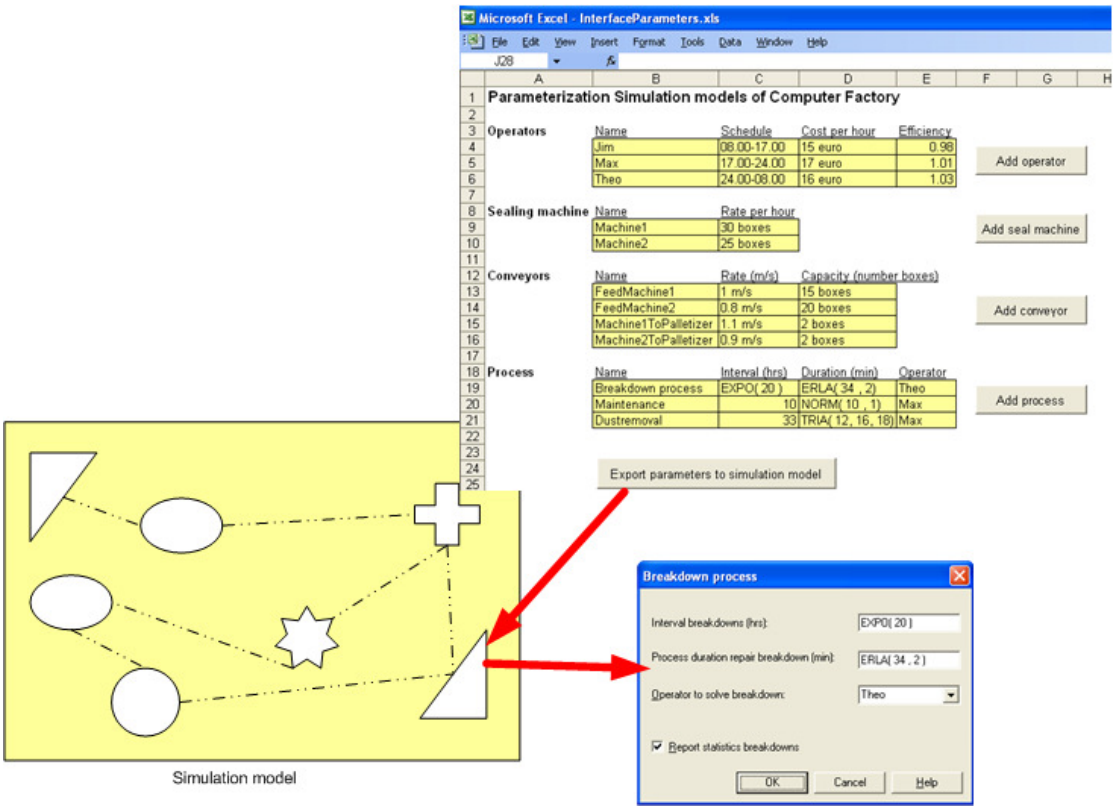


Figure 5.13: Example parameter setting instrument: spreadsheet enables adjusting all parameters of the simulation building blocks

5.5.3 Verify logic, data entry and syntaxes of simulation model before running

The generic simulation environments contain instruments to evaluate whether the syntax of a simulation model is correct. These instruments provide the model developer with feedback if parts of the model are not configured correctly, but they only check the basic features of a model like the number of parenthesis or whether all referred names are defined. The laboratory experiment clearly showed us that these checks are not sufficient. The generic simulation environment will provide feedback to model developers if something is wrong in the simulation model, but this occurs in a context that is aimed at the advanced model developer using the generic model constructs. A model developer with limited knowledge of the generic simulation environment often find it difficult to interpret generic error messages.

Model developers that use simulation building blocks need feedback that takes into account the building blocks level of abstraction. Figure 5.14 shows an example of feedback that could be provided if a process building block refers to an operator building block that is not instantiated in the simulation model. This instrument supports the model developer in parameterizing simulation building blocks (requirement 10) in the way that it is intended (requirement 2 and 6).

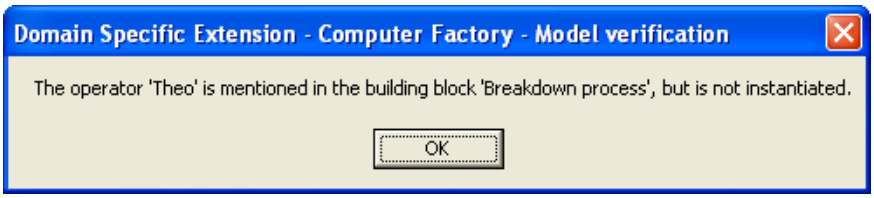


Figure 5.14: Example of result of automatic model check

Figure 5.14 is a customized message generated by the domain specific extension before the simulation run. In a generic simulation environment a model developer has infinite options as to how to model and represent a system. If a model developer starts using domain specific building blocks, then he or she no longer has infinite options. The structure of the simulation building blocks and the design decisions made by the developer of the set of building blocks limit the number of options. This limitation can be used to verify whether a model developer has followed the rules for the set of simulation building blocks.

5.5.4 Analyze output of the simulation model

Simulation models that are composed using a domain specific extension are easier to adjust. The case studies showed that ease of adjustment leads to a larger numbers of experiments being performed in the simulation study. Each of these experiments needs to be analyzed to determine the problem solution direction or the feasibility of a solution. Problem owners are not interested in viewing the statistics of one model construct or one type of statistics. They would like to see an appropriate combination of performance indicators, all generated and fed by different model constructs instantiated in the simulation model. In a support tool the total outcome representation of the simulation building blocks can be combined and thus better support the problem owner. Developing such an interface for an individual simulation model is a lot of work, but developing a generic solution based on the simulation building blocks of the domain specific simulation extension is an investment that can be spread over different simulation projects and thus be more cost effective. Further, a specific building block element can easily be used to gather and, in a structured way, provide the statistics to an interface. An example is shown in Figure 5.15, where an Excel sheet is fed with the output data of a domain specific extension.

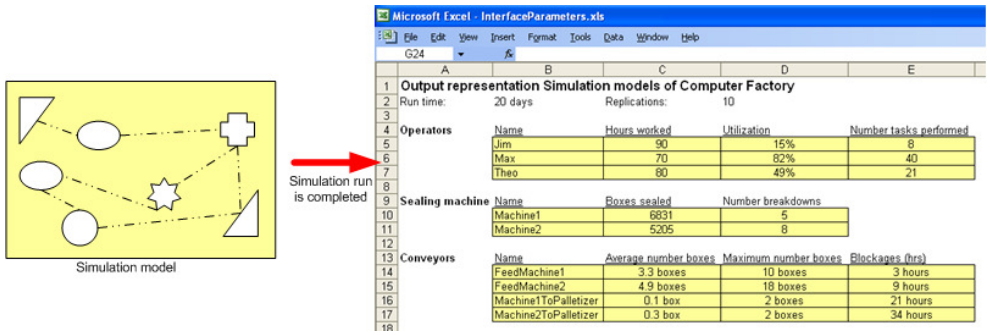


Figure 5.15: Example Excel sheet with output data of simulation run

5.6 Support and documentation for domain specific extensions

Developing a simulation model of a system is a complex activity. We do not expect that we ever will be able to develop a domain specific extension that will make the work so easy that model developers can perform this work without training or support. In the laboratory experiments we noticed that the training provided to the novices and the experts was sufficient to get them started, but not sufficient to let them solve all problems they encountered. We also noticed in the laboratory experiments that the experts wanted to find out more about the technical background of the model, while the novices did not have the background knowledge to do so. We concluded that the two types of model developers require different types of support to work comfortably with a domain specific extension.

The laboratory experiments showed that documentation is necessary: however, model developers will only read the manual if they get stuck or have questions about how to deal with certain system elements. When this happens they will not start reading the manual at page 1 and carefully read every page, instead they scan the document to find the section about their building block. There the model developer expects to find a clear explanation of the issue at hand, assumptions for the building block, and a small but clear example.

The documentation of the domain specific extension and the model constructs provided for the laboratory experiment focused mainly on providing an explanation of the input parameters. The difficulty that model developers encountered when attempting to solve their problems showed that this explanation was not sufficient. Model developers and potential users of the domain specific simulation should be provided with information that explains when to use the domain specific extension (**applicability**) and why the behavior of the simulation building blocks in a simulation model can be trusted (**trustworthiness**). Using the information for applicability and trustworthiness the model developers can decide whether a domain specific extension is suited for their purpose and whether they have faith that the domain specific extension will result in a valid simulation model.

The next step is to start using the domain specific extension. Model developers require support and documentation for this process, mainly covering individual simulation building blocks and building block elements in the domain specific extension to illustrate **usability**.

5.6.1 Support and documentation to illustrate applicability

A model developer who is considering using a domain specific extension and its simulation building blocks to represent a system needs to be able to verify whether the domain specific extension is suited for the type of systems that he/she wants to simulate. Developers of a domain specific extension do not aim to produce a product that can be used by every model developer for any

kind of systems. The developers model a domain specific extension and its simulation building block for a certain scope of systems. A good description of the scope is a first step to providing good documentation, but more support and documentation needs to be provided for the use by (future) model developers. This information should at least contain the following:

- Description of the scope of models that can be developed with the simulation building blocks of the domain specific extension.
- Overview of possible simulation experiments that are likely to be performed in the domain and that are supported by parameterization of the simulation building blocks or by instantiating new or different building block elements.
- Description of the level of detail of models that can be developed with the simulation building blocks.
- Outline of the assumptions that were made when building the domain specific extension, by defining the structure of the simulation building blocks.
- Overview of performance indicators that are generated by a simulation building block.
- Descriptions of small simulation models that show specific uses of the simulation building blocks.
- Example models developed using the simulation building blocks.
- Overview of expected influence of input parameters on the performance indicators, which explains the potential effect of the input parameters on the building block's performance.

5.6.2 Support and documentation to illustrate trustworthiness

Trust in a domain specific extension is hard to achieve and good documentation will only go part of the way to achieve it. Trust in a domain specific extension will be achieved by developers having faith in an extension and word of mouth advertisement by other users. Nevertheless, the trustworthiness of a domain specific extension can be influenced by providing documents and models that reflect the following items:

- Outcomes of experiments with small example simulation models.
- Descriptions of verification and validation steps to evaluate the behavior of the simulation building blocks and building block elements.
- List of model developers or problem owners that have used the domain specific extension.
- Success stories that explain why the model developers or problem owners used the domain specific extension.
- Background of the developers and designers of the domain specific extension.

5.6.3 Support and documentation to illustrate usability

The final type of support required is to help model developers who selected a domain specific extension to start using the simulation building blocks and the building block elements in the correct way. Below we list a number of points that should be addressed by the developers of a domain specific extension to help model developers to work with their domain specific extension:

- Process descriptions should show the model developer how to develop a simulation model with the simulation building blocks of the domain specific extension.
- Description of when to use which of the available building blocks elements for system elements.
- Description of the simulation building blocks which includes a description of the possible use, parameterization and performance indicators.
- Overview of warnings and errors that can be generated by the domain specific extension during the model development or experimentation with the simulation model; each error should contain suggestions how to solve the problem causing the error.
- Set of verification experiments with small simulation models to observe different system behavior using the same simulation building blocks.

The documentation provided with a simulation building block should enable a model developer to know when and how to use this simulation building block in a simulation model. The available documentation of a simulation building block is often used as reference material once a model developer cannot find out what to do with a building block or which building block to use to represent a system. This documentation of individual simulation building blocks should therefore be well structured, easy to understand and answer potential questions of the model developer adequately.

Also in other problem domains people are struggling to develop sufficient user documentation for building blocks. In the domain of software engineering several initiatives exist to define the minimum requirements for documentation of a software component. The aim of these attempts is to achieve standardization in documentation, to promote exchangeability, clarity and transferability of software components between different software developers (Verbraeck, 2002).

We analyzed three proposals for defining software components to judge their ability to also describe simulation model components. The analyzed proposals were the Web Service Description Language (WSDL) (www.daml.org/services/), a specification standard for business components developed by a working group of the German “Gesellschaft für Informatik”

(www.fachkomponenten.de) and an approach developed by Heisel and Souquières (2004).

The Web Service Description Language (WSDL) is designed (www.w3.org/TR/wsdl) to facilitate the use of services over the web. The goal of the WSDL is to provide better, unambiguous meanings of services and to support re-use of services or components by others, where the context of this re-use is not known in advance. The description of a web-service comprises three parts:

- the *service profile* for advertising and discovering services; it answers the question “What does the service require of the user(s), or other agents, and what does it provide for them?”
- the *process model*, which gives a detailed description of a service's operation, and answers the question “How does it work?”
- the *grounding*, which provides details of how to interoperate with a service, via messages, and answers the question “How is it used?”

In 1999, the working group “Component Oriented Business Application Systems” within the German “Gesellschaft für Informatik” started to develop a standard specification for business components (www.fachkomponenten.de). The result of this working group is a description of a business component, structured using seven levels. The purpose of business components is to support business processes. The *marketing level* describes business-organizational features of the business component and technical initial conditions. The *task level* contains the purpose of the business component and the tasks that it automates or supports. The functional terms of the business domain are explained on the *terminology level*. The *quality level* describes non-functional properties and quality features, and their corresponding measurement units and methods. The relationships between the services of the components and the cooperation with other components are specified on the *coordination level*. The *behavioral level* contains invariants and pre- and post-conditions. Finally, the *interface level* describes the different views of business components, services, parameters, data types and failure reports, as well as service signatures and assignment to business tasks.

Heisel and Souquières (2004) have developed a specification structure for software components that covers the functional aspects of such components. This structure is aimed at supporting those making the decision whether two components can be interfaced or not. The description of a software component consists of the specification of its export (supplied) interfaces, its import (required) interfaces, a usage protocol relating the export interfaces, and a relation between export and import interfaces. That relation states which export service relies on which import services.

None of the three descriptors fits completely with the need for simulation building blocks, but a combination of the documented work of all three workgroups has resulted in the list of items that can be used to describe of a simulation building block:

Name: a clear and representative name that enables model developers to grasp its intended purpose.

Objective: description why the building block should be part of the model. This could be to provide an overview of experiments that can be performed with models that are composed using the simulation building block.

Purpose: a natural language description of the building block should include a rough sketch of its functional behavior and thus of the function that the simulation model building block can fulfill in a simulation model.

Underlying assumptions: the underlying assumptions identify when a building block can be used and when it cannot be used. For example, if a traffic building block assumes that no blocking of crossings will occur, the simulation building block should not be used for situations where blocking plays a central role. The assumptions are often closely related to the overall objective of the building block.

Validation: validation and verification are one of the key issues when building and using a simulation model (Balci, 1997). Model validation makes sense only in the context of a specific purpose of the model and of the question to be answered using the model. Hence, using a simulation building block that is valid in a certain context does not guarantee that this building block is valid in another context. Therefore the validation item will have to show an overview of testing models and parameters that have been used to validate the behavior and representation of the simulation model component. Furthermore, case studies and problem descriptions that show the successful and unsuccessful use of the building block may be valuable.

Input and output ports including associated signatures: this is the technical interface description as identified in all software structures. It provides an overview of the triggers or messages the simulation building block can receive and will send to other building blocks. The associated signatures are an overview of the function or process started when something enters through an input port.

Parameterization: Which values of the component can/must be parameterized by the model developer? Parameterization information should also include the allowed ranges of values for parameterized attributes. Most often the model developer can parameterize the behavior and attributes of a simulation model component through a user interface of the simulation environment.

Expected effects of changes to parameters: the influence of parameterizable values should be clear to the simulation model developer.

Visualization / performance indicators: output generated by the simulation building block that provides insight into the behavior and state of the

component. Visualization mainly refers to insight during the simulation run. Examples of visualization are a console with messages or animations.

Description of the functionality: model developers will not use a simulation building block unless the behavior of the building block is absolutely clear to them. Model developers run the risk of inconsistencies in their model, different use of resources, different levels of abstraction, or a process flow that does not represent their system. For software components, pre and post conditions and usage protocols suffice, but in simulation model components, the functional behavior should be described in detail, preferably using a standardized and formalized description, like, for example, the DEVS-framework (Zeigler et al., 2000).

Illustrations of how it works: to support a faster decision whether or not to use a simulation building block, and an easier integration of a simulation building block into a composed model, the description of the functionality should be complemented by example models using the component, training material, or animation movies.

Examples provided to help the modeler should be small simulation models, preferably of a fictive and simplified system. The simulation model should be composed of a limited number of simulation building blocks, so it is easy to see the effects of an individual simulation building block on the overall system. These small models can also be used for demonstrating the effects of parameter settings to modelers. Assignments can be defined that use the original model as a starting point for adjustments to the settings of the parameters of the simulation building block. These assignments should preferably also provide outcomes of the adjusted model and an explanation of the alternative behavior of the simulation model. The box below shows an example for the domain specific extension “Computer factory”, which was introduced in chapter 2.

Example model “Breakdowns”. The simulation model shown in Figure 5.16 shows one sealing machine, a breakdown process and an operator. The sealing machine is waiting for computers in boxes, but in this simulation model the arrival of boxes is not included. Nevertheless the sealing machine has a breakdown every 24 hours, represented by the simulation building block “Breakdown process”. This building block will request an operator “Theo” to repair the machine. The operator requires on average 30 minutes to repair the machine. The output (right side of Figure 5.16) shows that after 240 hours there were 5.84 hours when the operator worked to solve machine breakdowns.

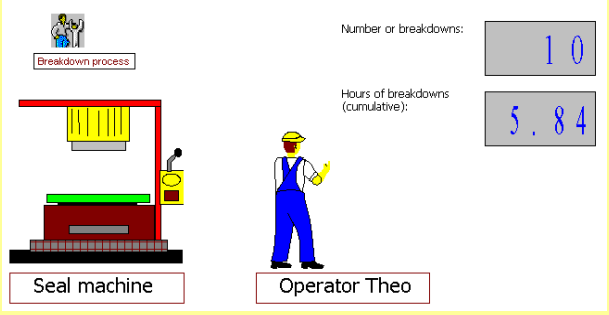


Figure 5.16: Implementation of simulation building block “Breakdown Process”

Training assignment for model "Breakdowns". Adjust the interval between breakdowns from an average of 24 hours to an average of 12 hours. The expected result is that much more time is spent on fixing breakdowns and that the number of breakdowns increases. Even though the decrease is only 50%, the number of breakdowns will not double, due to the time required to fix the breakdown. Figure 5.17 shows the change that should have been made in this assignment with the outcome of the simulation model.

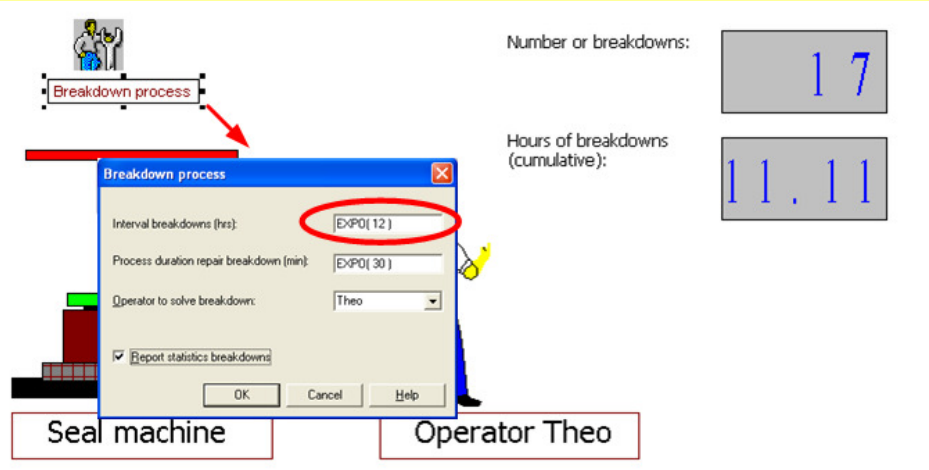


Figure 5.17: Adjusted simulation model with output of training assignment "Breakdown Process"

5.7 Design approach for improved domain specific extensions

In section 2.3.3 we introduced some steps for the development of a domain specific extension. We observed during the simulation studies, described in chapter 3, that we needed to extend the set of model constructs, to extend the performance indicators in the model constructs and to adjust the terminology to match the understanding of the model developers in the domains. In other words: we designed a set of model constructs that was not sufficiently linked to the domain, because system elements and important information were missing.

The initial phase of the design of the domain specific extension did not receive sufficient attention, and as a result we did not describe all required system elements. In this section we provide a design approach that pays more attention to setting the scope of the domain specific extension, which hopefully results in a set of simulation building blocks that better match the expectations of the problem owners in the domain.

In addition, the design approach introduced in section 2.3.3 was based on model constructs and not the concept of simulation building blocks. Therefore the design approach for development of domain specific extensions is extended with considerations regarding the simulation building block guidelines (section 5.4), the development of additional tools (section 5.5) and activities to result in documentation and support for model developers (section 5.6).

5.7.1 Roles

Developing a domain specific extension requires insight into a problem domain. The extension should not just be focused on one particular system, but on multiple systems. Developing a domain specific extension requires advanced knowledge of the field of simulation and of the selected generic simulation environment. Knowledge of the field of simulation is needed to support experiments, to enable analysis, and to provide validation. Advanced knowledge of the generic simulation environment is needed to develop simulation building blocks that are sufficiently parameterized.

It is very rare that a person knows everything about a domain and is also a simulation expert. Except for these rare cases we suggest that a team of experts should be formed, with different roles, to design a domain specific extension. One or more roles could be performed by one person, but we think that the quality of the final domain specific extension will be better if each role is performed by a different person, and some roles even by several persons. The roles that we have identified in the design of a domain specific extension are:

- *Domain expert / problem owner:* this role provides knowledge of the domain and a vision of the future demands for the simulation study to define the type of simulation models and experiments to be carried out.
- *Model developer*: this role will be performed by the users of the domain specific extension to develop a simulation model and to perform experiments with the simulation model. The knowledge of these persons can range from “have heard about simulation” to “have experience performing simulation studies”. In the laboratory experiment we found that both types of persons could use the simulation environment and that they needed different kinds of support.
- *developer domain specific extension:* this role will design the domain specific extension, implement the individual simulation building blocks and write the documentation about the building blocks and support tools of the domain specific extension.
- *tester domain specific extension*: this role will use the developed domain specific extension and the support tools and evaluate whether the functionalities mentioned in the documentation can be achieved. The tester will do this by developing small simulation models of representative systems in the domain.

5.7.2 Process steps and milestones

In Figure 2.11 is a process described to develop a domain specific extension. This process helped us to execute the case studies in chapter 3, but the input of our lessons learned and the identification of simulation

building blocks, use of additional tools and improvements to the support and documentation enable us to improve the process and define a prescriptive design approach for the development of domain specific extensions.

In this design approach the main focus is on the specification of the simulation building blocks. The extra attention in the design is aimed at avoiding that the implemented simulation building blocks and building block elements cannot represent the necessary system elements for the simulation studies. The design approach also includes the development of additional tools for model development, model parameterization, model verification and output representation to complete the domain specific extension. Finally, the domain specific extension should provide the support and documentation as described in the previous section.

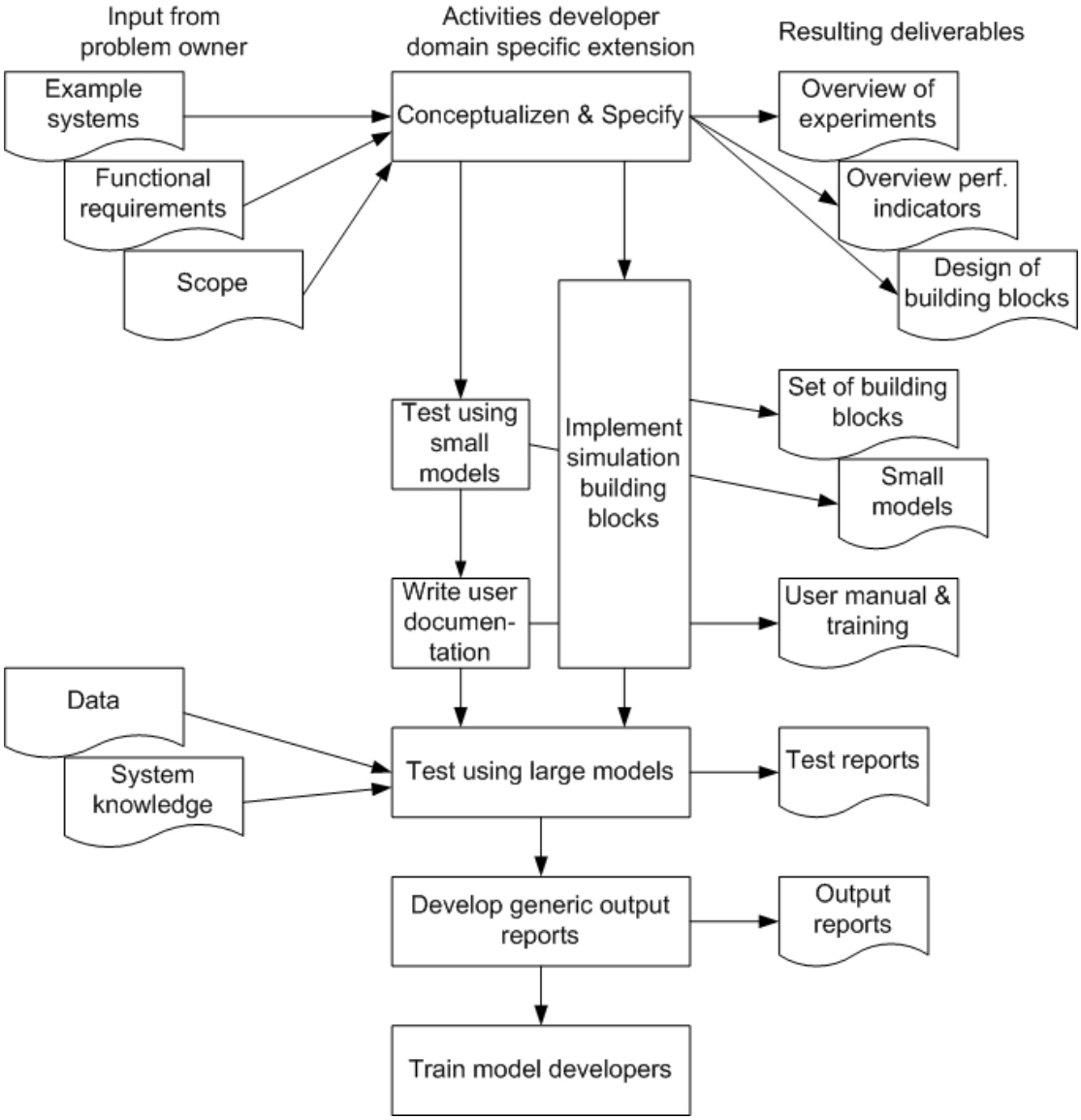


Figure 5.18: Overall process development domain specific extension

The overall process of designing a domain specific extension is shown in Figure 5.18. On the left we show the type of information that is provided by domain experts and problem owners. The process steps in the middle of this figure are carried out by either the developer or the tester of the domain specific simulation. These process steps then result in the products shown on the right of Figure 5.18.

This design approach described in Figure 5.18 does not describe how the problem owner or domain expert gathers the material they feed into the design process. This will depend on the size of the group of domain experts and their availability. A very efficient way is to bring all the problem owners together and let them provide their system knowledge. This could be done using a GDR session (De Vreede, 1995), but interviews or literature research into the problem domain can also be used to gather sufficient information to design a flexible domain specific extension. In this the 'example systems' are not existing sets of simulation models, but common domain knowledge the team is aware of and systems that should be suited to be simulated with the domain specific extension.

The first step in Figure 5.18 is 'Specification'. We concluded in section 5.2 that an extension of this process is needed to help developers design a domain specific extension that matches the structure of simulation building blocks. This process step is further explained in Figure 5.19 and is an improvement to step 1 (Object oriented and process oriented decomposition and abstraction) and step 2 (Generalize system elements) as introduced in the initial approach for development of domain specific extensions in section 2.3.3.

The other process steps in the design of a domain specific extension are fundamental to the development of a domain specific extension. They are listed below:

- 'Implementation of simulation building blocks and building block elements' is the process step to instantiate the building blocks as an extension to a generic simulation environment, based on the provided specification. Replaces step 3 (Instantiate system elements as domain specific model constructs) of section 2.3.3.
- 'User documentation' can be written while the simulation building blocks are being implemented. Extra step introduced to enable the development of support and user material as introduced in section 5.6.
- 'Test using small models' is the development of small models by a tester and the use of experiments to validate the behavior of the separate simulation building blocks. These models are used by the model developers to help them to understand how to use simulation building blocks (requirement 2 and 6). The model developers can also use these models to verify the exact behavior of simulation building

blocks (requirement 8). This was covered in step 4 (verify domain specific model constructs) of section 2.3.3, but add more activities and details of verifications to be performed.

- 'Test using large models' is the instantiation of the simulation building blocks and building block elements by a tester to represent the example case studies. These models can be used to demonstrate the capabilities of the building blocks in a full simulation study (requirement 1). This is an extra activity to increase the trustworthiness of the simulation building blocks. In section 2.3.3 this was not considered yet.
- 'Development of generic output reports' results in reports for the model developer that provide all the data that the model developer might need to evaluate their simulated system. This activity is added to support the development of additional tools as introduced in section 5.5.
- 'Training model developers' can be a formal training, or a self-study project in which the model developer uses the domain specific extension to gain understanding and trust in the provided simulation building blocks. This training satisfies requirements 2 and 6. Possibly a more technical training can be provided as well to train model developers in how to handle new building blocks and building block elements. This is an extension to support model developers as suggested in section 5.6.

5.7.3 Conceptualization and specification for a domain specific extension

The conceptualization and specification of a domain specific extension can be divided into four parts as is shown in Figure 5.19. 'Determine scope' deals with describing what the simulation models that will be instantiated using the simulation building blocks can represent. 'Decompose problem domain' is the decomposition of different systems of the problem domain into sub-systems and system elements, using the constructs view and functionalities view introduced in chapter 2. 'Design building blocks' is the transformation of system elements into building blocks, where some system elements might be combined or ignored to support flexible yet easy use. 'Design building block elements' is the final phase of the specification and is focused on to a more technical level, like providing a user-interface, and interfaces between building blocks.

Each of these phases of the specification is decomposed into process steps. The order of these process steps is the preferred order to develop the specification, but as any design process, the actual execution will be highly iterative and returning to earlier phases while design is going on. In the first phase (determine scope) the main information from the domain experts will be used. The problem domain is limited to a set of problems to be modeled. This set of problems will influence the selection of example systems that is used for

developing the domain specific extension. The aim of the example systems is to verify whether design decisions donot limit the model developer to make simulation models of the selected example systems. Model developers should be possible to construct simulation model of each of the selected example systems as much as possible using the simulation building blocks that will be developed in the domain specific extension. Therefore the descriptions of these example systems will be “requirements” for future design decisions. The developers of the domain specific extensions should evaluate in each design step whether they can still fulfill the requirements of the example systems. For example, an example system in the domain of passengers at airports could be the case study of KLM check-in. In this case study the type of passengers was important, thus one of the key-input parameters to be defined in the phase ‘Determine Scope’ should be “type of passenger” as was expected that similar processes apply to airports worldwide.

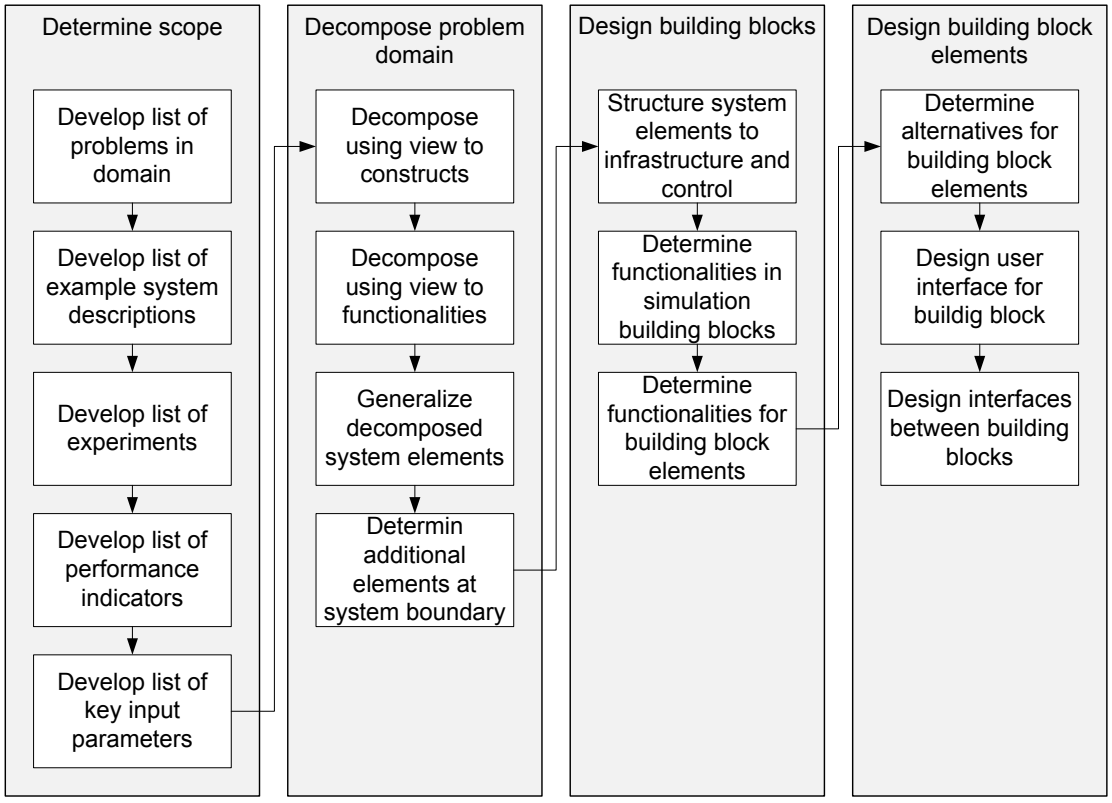


Figure 5.19: Conceptualization & specification for domain specific extension

The phase ‘Decompose Problem Domain’ is similar to the process steps 1 ‘Object oriented and process oriented decomposition and abstraction’ and 2 ‘Generalize system elements’ of the design approach introduced in section 2.3. The difference with the design approach described in chapter 2 is that the generalization is performed separately for the two views, i.e. the object oriented and process oriented view.

The specification phase ‘Design building blocks’ is the phase in which the objects that have been decomposed and generalized are allocated to simulation building blocks. Some of these objects will be simulation building blocks, others will be allocated as a functionality for one of the other simulation building blocks. The result of this phase strongly depends on the types of experiments that are foreseen and the importance of the objects that have been identified using the two different views on decomposition. Some of the trade-offs have to do with the actual decision whether to make something into a simulation building block or to allocate it as a functionality.

Finally, the specification of the domain specific extension focuses on the technical aspects. We have mentioned the importance of the interfaces of the simulation building blocks to enable the simulation building blocks to interact correctly. We also identified the need for the usability of building block elements to enhance their flexibility for model developers. How these two concepts are incorporated into the simulation building blocks and the domain specific extension depends partly on the selected generic simulation environment. A domain specific extension developed as an extension of a JAVA based simulation environment can use more of these software engineering concepts than a domain specific extension that is developed on top of procedural simulation environments like Arena or Witness.

5.8 Case studies to apply domain specific extensions

Section 5.4 (Simulation building blocks), section 5.5 (Additional tools for domain specific extensions), section 5.6 (Support and documentation for domain specific extensions) and section 5.7 (Design approach for improved domain specific extensions) describe the contribution of this dissertation to the field of discrete event simulation. This chapter provides concepts and approaches to overcome problems with simulation studies observed in literature, laboratory experiments, and case studies. In the laboratory experiments in chapter 4 we showed that domain specific extensions are more effective in simulation studies than generic simulation environments. In the following chapters (6, 7 and 8) we will show domain specific extensions that apply this contribution and see the effects in simulation studies that are performed based on the suggestions formulated in this chapter. Most important is that the domain specific extensions contain simulation building blocks and building block elements implemented using a generic simulation environment and following the simulation building blocks guidelines introduced in section 5.4. Further the domain specific extensions contain additional tools to support the model developer in creating simulation models and perform experiments according to the ideas described in sdection 5.5. The model developers and problem owners are supported with training and user documentation as mentioned in section 5.6. These domain specific extensions with their simulation building blocks, additional tools and support material are developed according to the design approach described in section 5.7.

The domain specific extensions described in the following chapters have resulted in successful simulation studies that have been performed fast and easy, benefiting from the concept of the simulation building blocks. We do not describe the simulation studies in detail, but focus mainly on the design of the domain specific extension, i.e. on the result of following the design approach and the 22 Simulation Building Block Guidelines. The development of the different domain specific extensions have been team efforts in which we were highly involved, in the execution of the simulation studies we have occupied the backseat and observed the execution and use by model developers. The observations are organized per case study and in chapter 9 they are combined to overall conclusions of the research contribution for domain specific extensions containing simulation building blocks, additional tools and documentation and support.

5.8.1 Case study for supply chains described in chapter 6

The first domain specific extension is for the domain of supply chains which will be used to prove that the design guidelines can be applied in different generic simulation environments (chapter 6). This case study answers research question 2A related to the difficulties to handle the unlimited freedom in modeling by the model developer:

> *What constructs and design approach will enable that domain specific extensions can be defined independent of the generic simulation environment in such a way that the model developer is supported, but not limited to one way of representing a system element?*

This case study will focus on the flexibility of simulation building blocks and building block elements to easily change the behavior of a system in a supply chain using existing mechanisms as well as customized mechanisms that were not part of the domain specific extension.

5.8.2 Case study for container terminals described in chapter 7

The second domain specific extension deals with the design of container terminals, where the focus of the case studies was to provide fast and easy insights into the suitability of a terminal design (chapter 7). This case study mainly addresses research question 2B related to the challenge that model developers need to be experts in multiple areas:

> *What methodologies, approaches and techniques can be offered to a model developer to support the use of domain specific extensions in the activities of a simulation study?*

The simulation building blocks represent a high level of abstraction and the domain specific extension contains additional tools to enable the model developers to quickly develop simulation models automatically from drawings

and database entries. The model developer can thus focus on the analysis of the automatically generated output. In this case study the complete process of a simulation study with a domain specific simulation extension (section 5.5) is automated.

5.8.3 Case study for Nestlé production facilities described in chapter 8

The case studies of the supply chains and container terminals have been mainly applied as a proof of concept. Though the problem was real, the problem owners were not. Therefore the third case study focuses on real problems with real problem owners who really need answers from a simulation study (chapter 8). The third case study explains how a dozen different simulation studies have been performed using a domain specific extension for Nestlé's production facilities. This case study gives insight in all aspects of the theory introduced in this chapter, and mainly answers research question 2C related to the challenge that model developers do not speak the language of the problem owner:

> *How can be ensured that the domain specific extension gets the model developer closer to the language of the problem owner?*

The Nestlé case is an ideal case study to demonstrate how the language gap was closed by using a domain specific extension, and how developments in the domain were translated into further development of the domain specific extension.

5.8.4 Way of describing the case studies

The three cases studies in chapter 6, 7 and 8 are all described in a similar way. Each case study has a short introduction of the domain that the domain specific extension should support, and it explains the reason why a domain specific extension should be developed. The case studies have all been selected for the fact that the domain specific extensions have been used several times and thus they needed to be adjusted and improved. Each case study focuses first on the initial development, by describing the team that developed the initial specification (scope, problem domain, define building blocks and define building block elements) following figure 5.19. All three case studies have hundreds of pages of documentation that elaborate on the specification, the implementation, and the user information of the domain specific extension. In this thesis only the key concepts, building blocks and their interfaces are described to provide a feeling for the design of the domain specific extension.

The implementation focuses first on the generic simulation environment that was used, followed by a description of the additional tools that have been developed and the (user) support that was provided by documentation and

training material. Again, this is not a complete account of all available documents, but an overview to show what was available for users and simulation experts who continued the development in later stages of the life cycle of the domain specific extensions.

The result of the development and testing of the domain specific extension are briefly discussed by evaluating the 22 Simulation Building Block Guidelines. In the evaluation of these guidelines, the extensions of the domain specific extensions are described as well.

Each case study section ends with a short description of the simulation studies performed with the domain specific extension and observations about the success or failure of the domain specific extension to support and enhance the simulation study.

5.8.5 Additional case studies not described in detail

In addition to the three main case studies, many more simulation studies using domain specific extensions were carried out. The most important of these simulation studies are described in appendix 1. The appendix provides an overview and short description of the domains and cases that were performed in the period of 2002 to 2007, but which are not discussed in detail in this thesis. In each of these cases a domain specific extensions has been developed and used in at least one simulation study within that domain. These cases are only briefly described.

6 Application to supply chains

6.1 Why develop a domain specific extension?

Supply chains are an important part of how organizations worldwide add value to products via storage, transfer and assembly. The costs related to activities in the supply chain, e.g. transport, procurement and storage can have a significant impact on the cost of the final product. In addition the availability and demand of products changes over time, resulting in alternative patterns of purchases and sales within the supply chain. The structuring of a supply chain, i.e. selection and cooperation between organizations, needs to be adapted to keep costs low and to optimize interaction between supply-chain partners (Boyson et al, 1999).

Current supply chains use real time exchange of data via internet and email. A customer can order his product via websites of different retailers and easily evaluate their offers to buy the same product from the cheapest retailer. Retailers on their turn also use the spot market, retrieving their stock replenishments from whoever provides the product at the lowest price. The key characteristics of a spot market are the placing of orders with different suppliers and the use of short term contracts (Cooper et al, 1997). The competition in this market is on price and availability of product. The actors have different mechanisms to manage their price and product availability for customers. Verbraeck (2004) describes how simulation models can be used to evaluate mechanisms for supply chain management and to help companies to decide on a mechanism for order generating, pricing and offer selection.

In a supply chain the actors exchange information. The key information exchanges are requests for quotations (RFQs) that an actor sends to several actors that are able to deliver the requested product(s). These actors will return a quote including price and delivery date. The actor that requested the product(s) will select which of the quotes of the supplying actors matches its requirements in the best way. The selected actor will receive an order to deliver the product(s) according to the conditions in the quote. The delivering actor will send a commitment, the products and a bill to get paid. This complete process, in particular between a customer and a supplier, is shown in Figure 6.1. All actors that participate in the supply chain will have their own way of handling the incoming messages, for example, a supplier might decide not to react to customers who are farther away than 2000 km. How the actors react to the messages and how they generate their own messages are key elements to represent in a simulation models of supply chains.

Simulation models are often used to design the structure of the supply chain and to evaluate alternatives like order size, order moments, use of external organizations and reaction times to demand. Supply chains have become an important topic with the ability of customers and organizations to easily shop worldwide and select the best product and delivery methods for

their requirements. Organizations that try to (re)design their supply chain network have different choices and mechanisms to react to demand. This requires models that can easily be adjusted and that can handle large amounts of data in combination with management decisions of allocation mechanisms to be used.

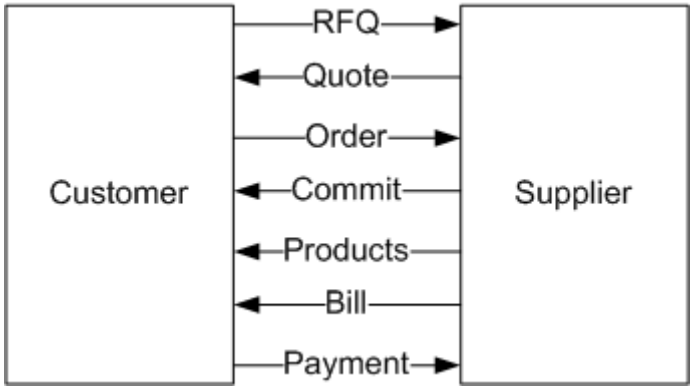


Figure 6.1: Process interaction between customer and supplier (Van der Hee, 2002)

The complexity of a supply chain network in combination with the vast amount of possibilities and information triggered the use of simulations. Van der Hee (2002) used a domain specific extension to define a set of supply chain teaching cases at the R.H. Smith Business School of the University of Maryland in the USA to illustrate different concepts of supply chains to the students. The simulation models that supported the teaching cases needed to be configurable and extendable by MBA students without any further training in the simulation environment, comparable with the complexity of the laboratory experiment described in chapter 4

A second use of the domain specific extension has been to evaluate the supply chain processes for turbine spare parts of the engines of the F101 fighter jet of the USA Department of Defense (Tewoldeberhan, 2005; Jacobs, 2005) where the focus was on regenerating the supply chain based on real time data.

6.2 Initial team to develop domain specific extension

The use of simulation models for supply chains has been one of the key research topics at the Supply Chain department of the R.H. Smith Business school of UMD. They have been supported by simulation modeling experts from Delft University of Technology (TU Delft) to jointly develop the domain specific extension. Jointly experts from these organizations have performed the specification of the simulation building blocks and the implementation in different simulation environments (Corver, 2001; Van der Hee, 2002; Tewoldeberhan, 2005; Jacobs, 2005; Van Houten, 2007).

The role of model developers was initially performed by the developers of the domain specific extensions themselves. The initial models were used by the supply chain experts to run the experiments.

6.3 Specification of domain specific extension

6.3.1 Scope

The scope of the simulation models is defined as different actors working together in a supply chain. The actors are generally defined as customers, retailers, distribution centers, manufacturers and suppliers. The scope of the supply chain can contain the whole chain from raw material until final product, but the actor concept also allows a focus on a part of the supply chain where the supplier provides semi finished products.

The interaction between the actors is in essence described in Figure 6.1. An actor needs a supplier for a product and the supplier provides that product given certain conditions are met. The potential complexity of the information exchange and the physical good exchange are endless and have been part of the initial scope of the design of the simulation building blocks.

The experiments, performance indicators and input parameters have also been strongly limited. The focus of the project was to support the teaching cases and thus enable experimentation with supply chain concepts such as make-to-order, make-to-stock, bull whip-effect and JIT-delivery (Van der Hee, 2002).

6.3.2 Problem domain

Defining the problem domain was a further identification of the processes and functionalities of the actors to be simulated in the scope of the domain specific extension. This resulted in a more precise definition of the types of interaction that the actors had. Based on the interaction patterns, we split an actor into a logical and a physical actor. The logical actor keeps track of stock, puts requests to other actors and keeps track of the reservations placed by other actors. The physical actor is responsible for preparing the goods (e.g. the manufacturer that requires a couple of days to make an order to the specification of the customer) and keeps the physical stock. The interaction also includes a yellow page object that is aware of the different actors in the system and knows which goods the actor offers. In this way the actors can easily identify other actors that are probably suited for their business needs.

Another example of identified new objects to handle the scope of the actors was the introduction of a transport actor that enabled the transport of goods from one actor to another actor.

6.3.3 Building blocks

The scope of the actors, the extension of the initial actors with the yellow pages and the transport actor were good a preparation for the set of building blocks. The separation in a logical and physical actor resulted in three families of building blocks: 1) logical actors; 2) physical actors; 3) support actor like yellow pages and transport actor.

6.3.4 Building block elements

The building block elements definition of the logical and the physical actor also came to life reasonably simple. Each process step in the interaction scheme of Figure 6.1 resulted in a building block element, because for each of the interaction steps alternative implementations had been defined as part of the experiments in the scoping and the configuration of the key input parameters. The way an actor handles an incoming document or message defines its way of acting, and turns the overall behavior of the actor into e.g., make-to-order or make-to-stock. For example, a building block element was defined for the functionality ‘determine to reorder’. In a make-to-stock concept the ‘determine to reorder’-building block element analyzes the quantity still in stock, while the make-to-order concept was represented by a building block with a ‘determine to reorder’-building block element that keeps track of the incoming orders.

The logical actor and the physical actor received different building block elements to represent their functionalities in the process of ordering. For example the logical actor client, retailer, distribution center and manufacturer all possess the building block elements ‘order generator’ and ‘yellow page selector’ among many others. In Table 6.1 these two building block elements are listed with a number of their alternatives. The other building block elements and their (implemented) variants are well described by Corver (2001) and Van der Hee (2002).

Table 6.1: Example of functions and possible building block elements to represent behavior logical manufacturer

Function	Building block elements	Description
Order generator	Standard	Generate a new order for materials or products at regular times for a constant quantity.
	Periodic	Generate a new order for materials or products at regular times, but the quantity differs per period of the year.
	Continuous	Generate a new order for materials or products at the moment that the stock quantity is below an identified level.
	Customer	Generate a new order for materials or products without looking at any stock levels, just according to a random distribution for a time between requests.
	EOQ	Generate a new order for materials or products, depending on calculations for the economic order quantity that is repeated once every couple of months.
Yellow pages selector	Standard	Make a shortlist of potential suppliers for desired materials or products.
	Distance	Make a shortlist of potential suppliers for desired materials or products that are a maximum distance from the requesting actor.
	Type of Actor	Make a shortlist of potential suppliers for desired materials or products that are of a certain type of actor.

6.4 Implementation

6.4.1 Extension of simulation environment

A special aspect of this domain specific extension is that the building blocks and building block elements have been implemented by different developers in three different generic simulation environments. The implementation in the generic simulation environment eM-Plant was the first completed version, developed to enable validation of the concept of the building blocks and the use of building block elements to represent the information flow in supply chains (Corver, 2001). The second implementation was in the generic simulation environment Arena, aimed at use by none modeling experts of the R.H. Smith school to increase their knowledge of the concepts of supply chains with practical tools (Van der Hee, 2002). The third implementation was based on Java objects in the DSOL library, aimed at distributed modeling of large and distributed supply chain networks (Tewoldeberhan, 2005; Jacobs, 2005; Van Houten, 2007).

Development in eM-Plant

The strong point of the domain specific extension in eM-Plant was the ability to apply object oriented programming for the simulation building blocks, but offer the model developer to overrule defined inheritance and freely read and write attributes of other objects. As a result the building blocks and the building block elements could rapidly be developed and the design made by

Corver (2001) could be implemented in a straightforward way. The disadvantage of the generic simulation environment eM-Plant was mainly the availability of licenses that hindered the use of the domain specific extension beyond pilot stage, for example to support the teaching cases.

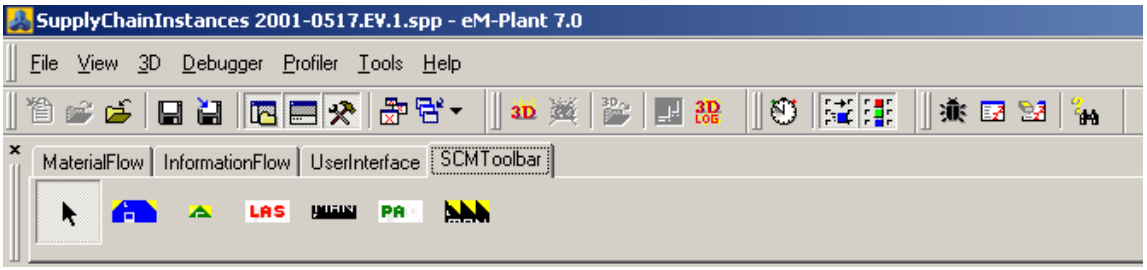


Figure 6.2: Families of building blocks in eM-Plant

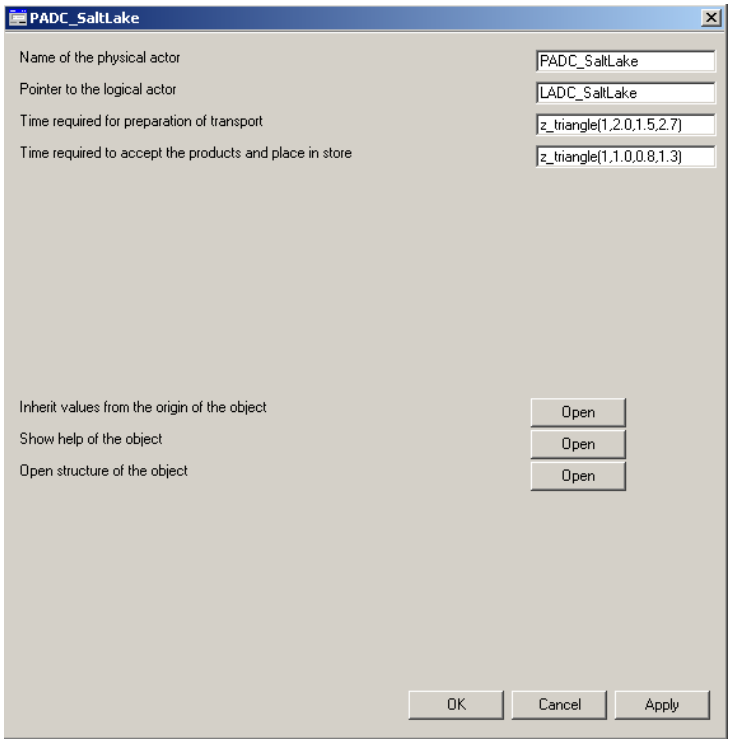


Figure 6.3: Information for the Physical Actor (DC) (Corver 2001, p51)

The implemented simulation building blocks were combined in a specific Supply Chain toolbar as an extension to the generic modules in the simulation environment eM-Plant, see Figure 6.2. Each building block contained a user interface where the parameters of the building block and the selection of the available building block elements could be set by the model developer. Figure 6.3 is a screenshot of the simulation building block Distribution Center, the physical actor configured for a model of a supply chain in the USA.

The model developer using the domain specific extension in eM-Plant has the opportunity to bypass the user interface and manually adjust the mechanism in the building block by replacing a building block element or even override the logic of a specific method in the building block. Figure 6.4 shows

the inside of the manufacturer building block with its default building block elements that a model developer could adjust.

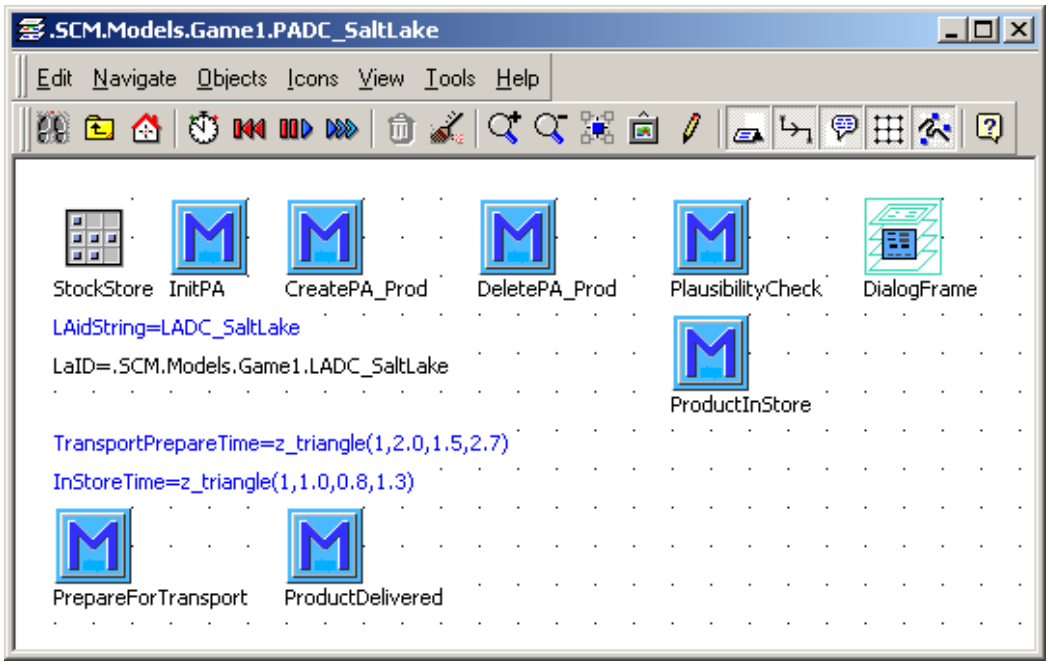


Figure 6.4: Building block elements in simulation building block based on eM-Plant

The status of the actors in the simulation model was maintained using variables only available within the building block or the related building block elements. Orders, RFQs and other information exchange were modeled via entities that contained attributes such as their origin, product type and amount. Gathering information from one or more different actors was handled via method calls via predefined interfaces and the information was stored within one or more entities that kept track of the status of an order (Corver, 2001). The information exchange between building blocks was always organized via a 'mail box' within the building block to enable flexibility of exchange of building block elements and not to reprogram the inner-working of different building blocks. Figure 6.5 shows the conceptual working of the post box in the building blocks of eM-Plant.

```
Method receive
trigger
If BBE=1 then
  Call X
Elseif BBE=2 then
  Call Y
Else
  Show Error
```

Figure 6.5: Logic in a method of eM-Plant to determine which building block element to trigger

Development in Arena

The main challenge of the domain specific extension using the generic simulation environment Arena was the flow concept that Arena applies and the lack of data types and data structures (Van der Hee, 2002). The concept of a building block that exists of several sub building block elements with object references as applied in the domain specific extensions developed in eM-Plant was not available in Arena. The solution was found in defining building blocks and separate building block elements that are connected by the model developer. Figure 6.6 shows at the right hand side an instantiated building block of a logical manufacturer and the triangles illustrate the need to link required building block elements, which can be dragged from the library at the left-hand side and connected to the triangles. In Figure 6.6 two building block elements have already been connected; the others still have to be done.

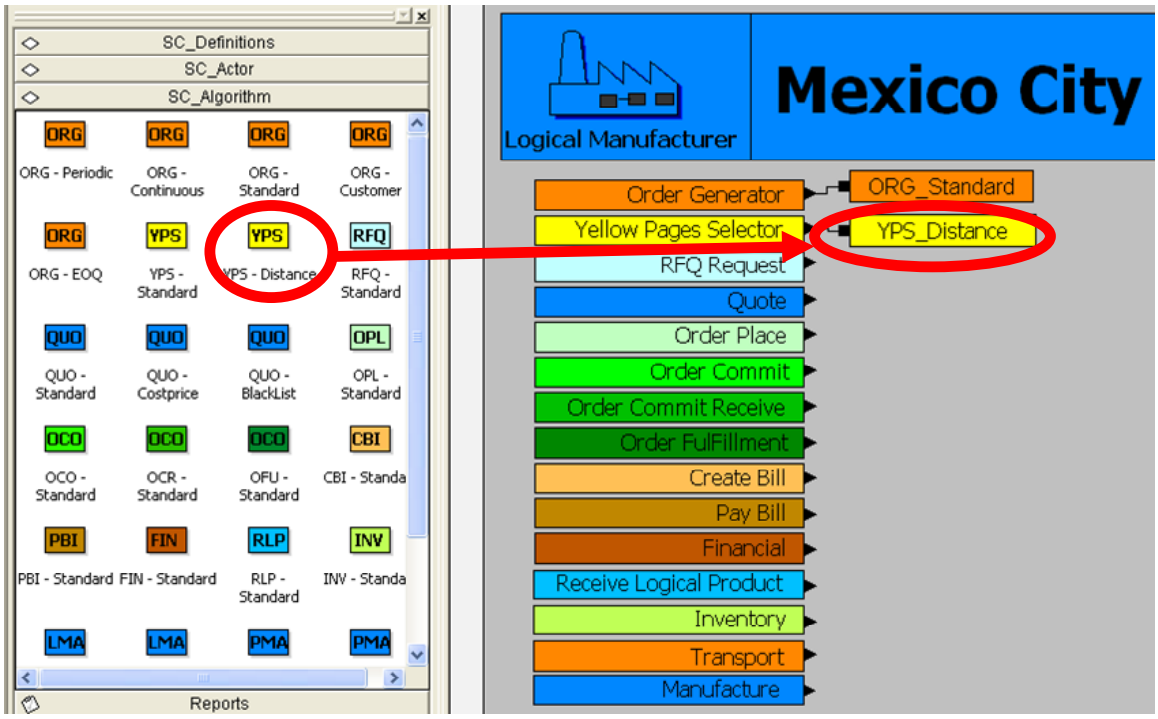


Figure 6.6: Building block elements in Domain specific extension based on Arena

The model developer has the option to select from all individual building block elements that exist, for example the top row in Figure 6.6 shows four different order generators implemented in the domain specific extension. Each of these building block elements have their own features and parameterization capability (see Figure 6.7), which at the same time protects the model developer from messing with the underlying logic and information.

As said before, one of the main challenges in the development of the domain specific extension on top of Arena was the lack of data structures. All information exchange had to be managed by entities that were sent back and forth between building block elements where the status had to be monitored using numbers (Arena has no strings). This made tracking difficult, but also

resulted in blocking of opportunities for model developers to enhance the functionality of building block elements with user defined logic developed with Arena generic model constructs.

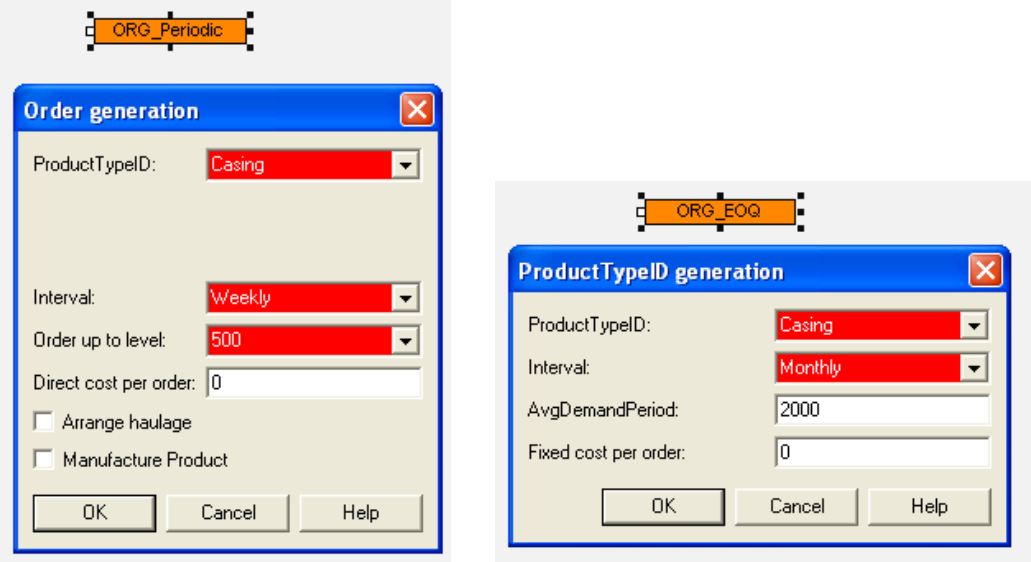


Figure 6.7: User interface for parameterization of two different order generator building block elements.

The implementation of the exchange of entities between building blocks and building block elements was achieved by defining dedicated stations where the entities arrive. The entities then follow one or more decisions based on their attribute values to be routed to the correct building block element. Figure 6.8 shows a part of the mechanism for a logical supplier actor. The left block evaluates which building block element has to be called. The bottom-right part evaluates which internal function of the simulation building block has to be called. If an entity arrives that does not match any of the defined building block element interface specifications then an error will be generated to warn the model developer of incorrectly defined message exchange between actors.

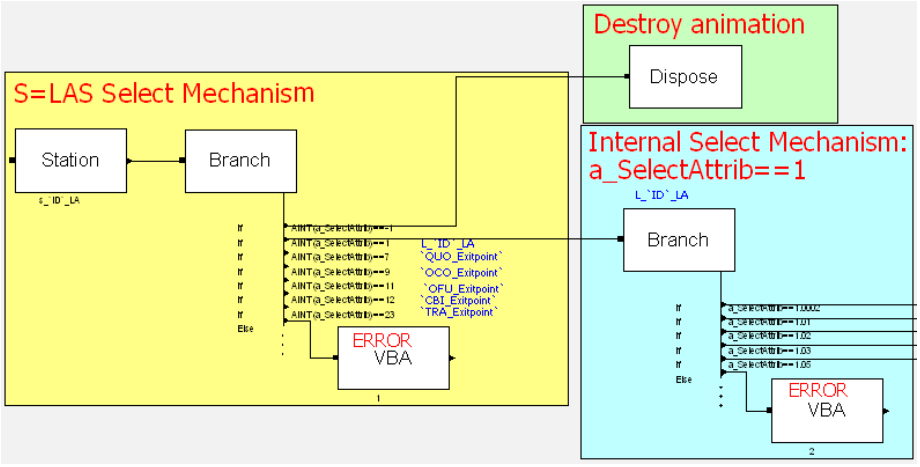


Figure 6.8: Internal logic in Arena to determine which building block element to trigger

Development in DSOL

The challenge in the case study for the USA Department of Defense was that information should be kept local to actors, and not be globally available in the model, due to the fact that the models needed to be distributed over the Web (Jacobs, 2005). The models should be easy to distribute to participants in worldwide locations and should run on computers, laptops and even PDAs that cannot install complete simulation environments like eM-Plant or Arena. The model of the supply chain should therefore not be one monolithic simulation model, but each actor (or group of actors) should be separate. In addition, the models of the supply chains should be predictive, but also represent actual events in history and forecast from there. Therefore, the domain specific extensions that have been developed on top of eM-Plant and Arena could not be applied as these generic simulation environments are not capable of retrieving a base situation from external sources (Jacobs, 2005).

The solution was found in the use of a new Java-based simulation engine that could be extended with the designed building blocks and the processes of the logical actors. The simulation objects of the DSOL library developed in Java have been designed especially to accommodate the capability of separate compositions of simulation building blocks connected into a simulation model with one combined simulation clock. The implemented building blocks worked much like the eM-Plant building blocks, using object oriented concepts, but their way of exchanging information was monitored in a more strict manner.

The use of both history and forecasting was implemented by two alternative instances of algorithms for the simulation building block elements. For example, the order request generation element of the logical actors either used information from historic data files, or prediction algorithms with economic order quantity.

Jacobs (2005) describes how the simulation model data is retrieved via a database connection, and results in instantiation of objects specific for the modeling of this supply chain as a DoDTrader and a Base, which are specifications of the building blocks Yellow Pages and Customer. The simulation models that use the domain specific extension on top of DSOL are composed using databases that contain structural information on the actors.

The result of the use of the domain specific extension on top of DSOL with the appropriate data sources is a simulation model that can be instantiated via any web browser or via a specific viewer application. The Java program will instantiate a map and use information from the specific databases to instantiate actors on their geographical locations.

6.4.2 Additional tools

The domain specific extensions developed using the generic simulation environments eM-Plant and DSOL have not reached beyond the state of proof-of-concept. The domain specific extension developed using Arena has

been used for all kinds of experiments and several developed simulation models. A large effort has therefore been put in developing additional tools. One to support the model development (mainly to overcome the labor-intensive job to instantiate all building block elements) and one to enable analyzing the outcome of the simulation model and to easily perform cross checks of the performance of all actors.

The model development in Arena has been supported by a Visual Basic application called “Builder Tool”. This allows a new actor to be added by selecting from pull down lists which building block elements are included. The actor will then be instantiated in the simulation model including all the selected building block elements. Figure 6.9 shows an example configuration of a new retailer with the name ‘New York’ that will be added to the simulation model with only standard building block elements for each function.

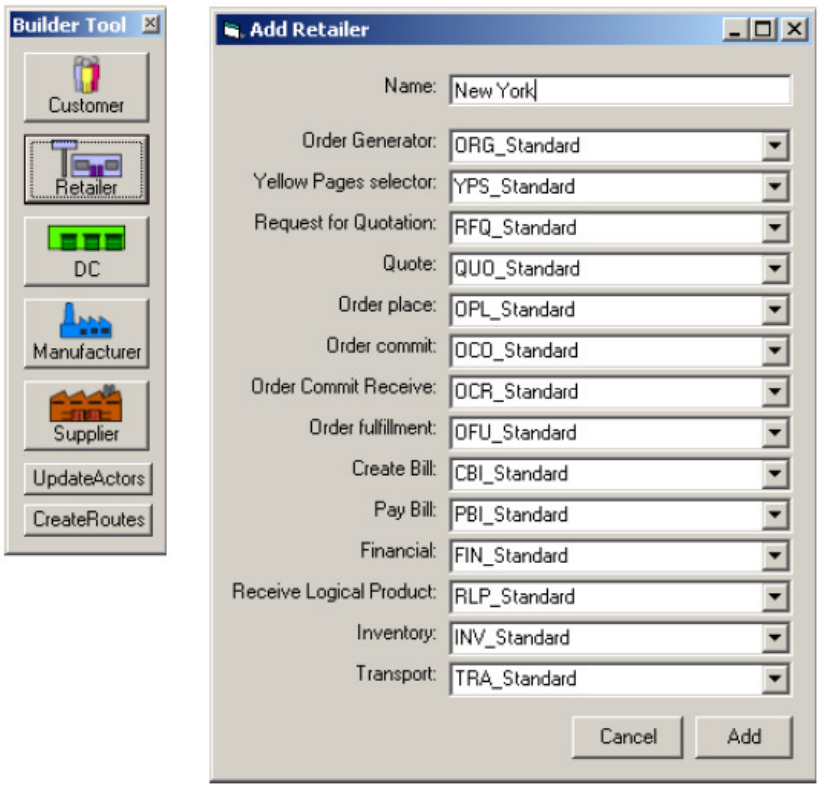


Figure 6.9: Tool of domain specific extension to instantiate logical actor in simulation model

The statistics of the simulation models developed in Arena have been collected in an MS Access database, developed especially to create reports of the performance of individual actors and compare the actors over time with each other. The main focus of the reports developed in the MS Access database was on the stock levels of the actors and on comparing out-of-stock moments based on parameterization of the building block elements of an actor in the supply chain (van der Hee, 2002). Figure 6.10 provides an example of a part of the report that can be produced by the MS Access database after the executed simulation model had run to completion.

InventoryQuery : Select Query

Actor	ProductType	OrderedLevel	CurrentLevel	ClaimedLevel	WaitingLevel	Date	Time
Seattle	Personal Computer	0	46	0	0	06-01-2002	18:16:00
Mexico City	Personal Computer	0	278	34	0	06-01-2002	18:33:00
Mexico City	Personal Computer	0	278	0	0	06-01-2002	18:33:00
Salt Lake City	Personal Computer	34	216	0	0	06-01-2002	19:03:00
Washington D.C.	Personal Computer	0	48	4	0	06-01-2002	19:15:00
Washington D.C.	Personal Computer	0	48	0	0	06-01-2002	19:15:00
Seattle	Personal Computer	0	43	3	0	06-01-2002	20:11:00
Seattle	Personal Computer	0	43	0	0	06-01-2002	20:11:00
Washington D.C.	Personal Computer	0	40	8	0	06-01-2002	22:15:00
Washington D.C.	Personal Computer	0	40	0	0	06-01-2002	22:15:00
Seattle	Personal Computer	0	39	4	0	06-01-2002	23:56:00
Seattle	Personal Computer	0	39	0	0	06-01-2002	23:56:00
Mexico City	Personal Computer	100	278	0	0	07-01-2002	00:00:00
Salt Lake City	Personal Computer	34	213	3	0	07-01-2002	03:01:00
Salt Lake City	Personal Computer	34	213	0	0	07-01-2002	03:01:00
Frankfort	Personal Computer	50	159	41	0	07-01-2002	03:01:00
Frankfort	Personal Computer	50	159	0	0	07-01-2002	03:01:00
Mexico City	Personal Computer	100	259	19	0	07-01-2002	03:01:00
Mexico City	Personal Computer	100	259	0	0	07-01-2002	03:01:00
Frankfort	Personal Computer	50	99	60	0	07-01-2002	03:01:00
Frankfort	Personal Computer	50	99	0	0	07-01-2002	03:01:00
Salt Lake City	Personal Computer	34	172	41	0	07-01-2002	03:01:00
Salt Lake City	Personal Computer	34	172	0	0	07-01-2002	03:01:00
Miami	Personal Computer	0	54	7	0	07-01-2002	03:05:00
Miami	Personal Computer	0	54	0	0	07-01-2002	03:05:00
New York	Personal Computer	0	40	4	0	07-01-2002	03:13:00

Record: 259 of 367

Figure 6.10: Example report generated in MS Access after simulation model run of domain specific extension

6.4.3 Support to users

The support to the users of the domain specific extensions developed in eM-Plant and DSOL has been limited, as it was just a proof-of-concept. The support to gain confidence in the building blocks and improving the applicability for a simulation study was not necessary for this reason, as the aim of the development and the use was to find out whether the building blocks could be used at all. Furthermore, the designer of the building blocks and the building block elements was the main user of the implemented domain specific extension in eM-Plant, thus fully aware of what was possible and which building block or building block element should be used in a particular scenario.

The domain specific extension developed in Arena has been used in classes of students at the R.H. Smith School of Business of the University of Maryland. The students who worked with the teaching cases developed in Arena also did not need a lot of convincing that the simulation building blocks were suited for their problems. They primarily needed support to understand how to interpret the results of the experiments and configure the building block and building block elements to improve the supply chain system. Their basic knowledge consisted of supply chain concepts. In two additional classes they learned how to interpret the effects of using supply chain concepts on the outcome of the simulation models. In these two classes they also received a demonstration how to make adjustments and use the additional tools to easily configure their simulation models to represent the desired system.

All students received a CD that contained example models, the additional tools and documentation about the building blocks and the building block elements. Step-wise they were guided through assignments using existing simulation models. The final course assignment was to optimize an existing supply chain by setting parameters and selecting the most appropriate building block elements. All this support material was sufficient to get the students working and perform the necessary experiments.

6.5 Use of Simulation Building Block Guidelines

This sub-section uses examples of the building blocks and building block elements from the three developed domain specific extensions. The design of the building blocks and the building block elements have based on the same conceptual model for all three implementations. Only the technical implementation differ. The main technical details are already described in section 6.4.1, therefore the Simulation Building Block Guidelines are only illustrated with examples of one of the domain specific extensions. The fact that the other environments are not mentioned does not mean that the guideline does not apply for the other environments as well, unless explicitly mentioned.

Guidelines related to self-contained building blocks

Simulation Building Block Guideline 1: data belonging to a building block should not be accessed by other building blocks directly, but only via defined interfaces.

Each of the actor building blocks contained its own information and state of its orders. For example, if a customer logical actor wants price information of a product from a retailer, it sends the retailer a request for quotation (RFQ). Figure 6.1 shows the complete set of messages exchanged between a logical customer actor and a logical retailer actor. Figure 6.8 shows how the interaction is technically achieved in Arena.

The messages shown in Figure 6.1 are the only way to exchange information in the domain specific extension. Similar interaction exists between retailers and distribution centers, between distribution centers and manufacturers and between manufacturers and supplier actors. None of the actors can retrieve information of another actor without the actor actively sending that information via a message.

Simulation Building Block Guideline 2: a simulation building block consists of a core and building block elements to represent functions and services.

Logical actors and physical actors have different functions. Logical actors are dealing with inventory levels, reservations of customers and price determination. The physical actor is concerned with the physical products that belong to an order, e.g. order picking, physical storage and the physical manufacturing process. Each of these functions is represented by a building

block element. Figure 6.6 shows all the building block elements that are required to represent the services of a logical manufacturer.

Simulation Building Block Guideline 3: data belonging to a building block element can be accessed by other building blocks elements of that building block without using the interfaces of the simulation building block.

The inventory level of an actor is kept by the building block element 'Inventory' of the logical actors. This building block element contains information of the current number of items in stock, expected deliveries of incoming stock and committed supply of products. This information is evaluated by building block elements like the order generation, i.e. generate a new order if the inventory is below a certain level, or the order commit, i.e. commit an order if stock is in place.

In the eM-Plant implementation this information is structured as separated object variables, e.g. *Actor.Inventory.ProductY.Stock* or *Actor.Inventory.ProductY.ExpectedDelivery*.

Guidelines related to interoperability of building blocks

Simulation Building Block Guideline 4: system elements that appear in different variants and processes in a system are represented by a family of building blocks and building block elements.

The logical actors belong to the same family, for example the difference between a manufacturer logical actor and a distribution center logical actor is the existence of the building block element 'manufacture', which triggers the physical manufacturing process. In the implementation in the generic simulation environment Arena, the building block elements for the logical and physical actors have been combined in one library, but the families of the building block elements can still be recognized thanks to the naming and shape of the building block elements.

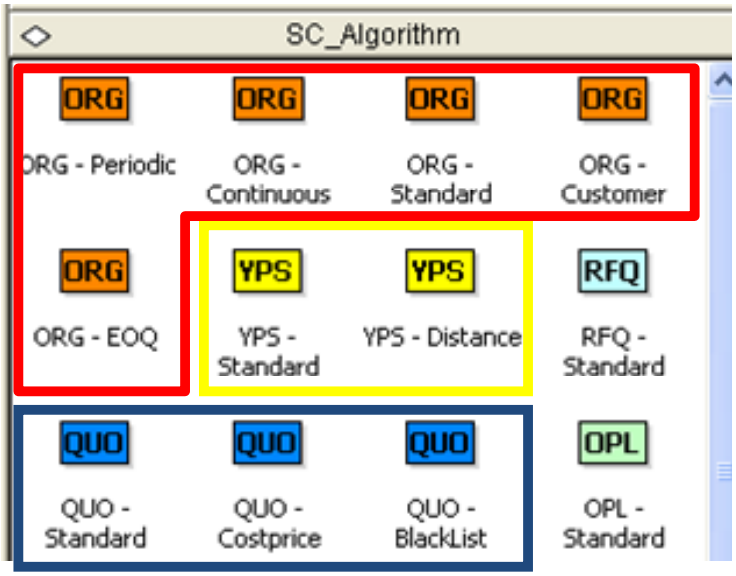


Figure 6.11: Families of building block elements of logical actors

Figure 6.11 shows the implementation of the building block elements with their family naming in Arena. The first five icons starting with 'ORG' are the building block elements for order generation, the two icons at the second row starting with 'YPS' are part of the family of Yellow Page building block elements and the three building block elements at the bottom of Figure 6.11 starting with 'QUO' are icons of the building block element 'Quote' as the reply to a request for quotation.

Simulation Building Block Guideline 5: building blocks are of different types, most common to have building blocks for infrastructure and for control.

The infrastructure in the domain specific extension is represented by the physical actors. The control is provided by the logical actors and its building block elements. The logical actor exchanges information with other logical actors until an order is accepted or manufacturing is triggered. The logical actor will then trigger the process that is part of sending a physical shipment or starting a manufacturing process at the physical actor. The physical actor will return a message to the logical actor when an order has been shipped or received or when the manufacturing process is finished.

Simulation Building Block Guideline 6: complex control mechanisms should be represented using control building blocks linked together to represent a flow.

The control processes in the domain specific extension for the supply chain are reduced to individual steps. The sequence of an order is not adjustable, and there is one process for handling orders as shown in Figure 6.1. How each step is handled is flexible using the building block elements. We decided not to use process representation for the control processes, because the type of experiments did not suggest a need to change the sequence in the order handling process.

Simulation Building Block Guideline 7: building blocks should be aware of each other's existence within a range of applicability.

Building blocks are linked in the implemented simulation models in different ways. The first is a fixed link between building blocks that are part of the simulation model. The second are real-time links between building blocks, depending on the state of the simulation model.

Examples of the fixed links are the connections between a logical actor and a physical actor, which is defined in the user interface of the building block, see Figure 6.3 where the physical distribution center in Salt Lake is connected to the logical distribution center 'Salt Lake'. A second example is the structure of the yellow pages object. One of the variants of the yellow page book is locally organized, whereby the logical actor registers itself to a local yellow pages building block, instead of to a globally organized yellow pages building block.

Examples of real-time links between building blocks are the negotiation and the execution of an order. Via the yellow pages, Request-For-Quotation and order placement, two or more actors are connected to exchange goods. The real-time link is a one-time connection not to be reused automatically because of the market situation in the supply chain.

Guidelines related to replaceable unit of building blocks

Simulation Building Block Guideline 8: extension of a domain specific extension can be achieved by introducing new building block elements for existing simulation building blocks.

Table 6.1 contains building block elements that have been identified by Corver (2001) as part of the domain specific extension to accommodate the type of experiments foreseen. Figure 6.11 shows that only a couple of these building block elements have been implemented in the domain specific extension on top of Arena. The developer of the domain specific extension can easily duplicate one of the existing building block elements and make the necessary adjustments to represent the missing building block elements. Van der Hee (2002) decided due to time constraints and the content of the teaching cases, that the development of the additional building block elements was not necessary. He provided sufficient technical descriptions to enable an experienced user of the generic simulation environment Arena to develop the extension of the domain specific extension in the same way that he extended the domain specific extension to contain the set of building block elements it currently contains.

Simulation Building Block Guideline 9: simulation building blocks and building block elements of the same family follow the same interface requirements.

The concept of interfaces illustrated in Figure 6.5 for the simulation environment eM-Plant and Figure 6.8 for the simulation environment Arena is applied for all building blocks and building block elements. The coding is kept the same for building block elements of the same type, thus the order generators ‘Standard’ and ‘EOQ’ use the same code numbers to refer the method call or entity to the correct functionality.

Guidelines related to encapsulating internal structure of building blocks

Simulation Building Block Guideline 10: simulation building blocks hide inner working.

The inner working of the building blocks in the domain specific extension developed in eM-Plant are hidden, but not made unavailable for the model developer, as can be seen in Figure 6.3 with the button ‘Open’ for ‘Open structure of the object’ to view the building block elements and even the methods of the building block elements.

The inner working of the building blocks in the simulation environment Arena has been hidden better than in eM-Plant. There is no way that a regular model developer can see what occurs within the building blocks.

Simulation Building Block Guideline 11: advanced model developers have to be able to unhide the inner logic and see how the processes and attributes are implemented.

The same text as mentioned for Simulation Building Block Guideline 10 could be used to explain how this guideline is achieved. In eM-Plant the model developer is discouraged, but can have a look. In Arena no option is available as the user interface of simulation building blocks developed in the simulation environment Arena cannot be overridden to show the internal code. Demonstrating the inner working of simulation models developed in the simulation environment Arena is done via documentation and process flows (Van der Hee, 2002).

Guidelines related to providing useful services or functionality of building blocks

Simulation Building Block Guideline 12: system elements should be represented by building block elements that can be extended with custom instantiations of model constructs of a generic simulation environment.

The building block elements in the Arena simulation environment could easily be connected with generic model constructs of this generic simulation environment, for example via the connection between the building block and the building block element. However, the entities that are sent over that connection are coded in a certain way and their attributes have values that are focusing on a particular function. There is a large risk that a model developer adds generic model constructs which alter the status of the entity. The building block element is then no longer capable of handling the function as intended. Furthermore, the additional tools that have been developed around the domain specific extension on top of Arena might not function due to the extra model constructs.

The advice is therefore not to tamper with generic model constructs in the domain specific extension developed in Arena. The disadvantage of lack of flexibility for the model developer should be matched by the flexibility to replace building block elements or to extend the set of building block elements.

Simulation Building Block Guideline 13: a building block can connect to model constructs of a generic simulation environment.

The same applies for this guideline as for Simulation Building Block Guideline 12.

Guidelines related to precisely defined interfaces for building blocks

Simulation Building Block Guideline 14: the model developer has to adjust the parameters of a simulation building block via a user interface.

Figure 6.3 and Figure 6.7 show examples of parameterization of the building blocks or building block elements in the simulation environments eM-Plant and Arena. These user interfaces are made available to the parameterization of all building blocks and building block elements in the domain specific extensions.

Simulation Building Block Guideline 15: use of domain terminology in the user interface provides insight in the suitability of a building block for a certain purpose and the meaning of its parameters.

The user interfaces shown in Figure 6.3 and Figure 6.7 are dedicated for the domain specific extension and contain a textual explanation of what the value represents. In addition it is described in detail in the user manual of the domain specific extension on top of Arena what the effect of changing values is (van der Hee, 2001). For example, increasing the quantity for evaluating the stock in the building block ‘Order Generation’ will result in higher safety stock and thus a lower probability of an out-of-stock event.

Simulation Building Block Guideline 16: parameters in a user interface of a simulation building block have to be checked for validity of the values.

At the moment that the OK button is pressed in the user interface of the simulation building blocks, checks are performed for the values entered in the fields. For example, in the implementation in eM-Plant it is evaluated whether the pointer of a logic actor to a physical actor is valid and in the implementation of Arena it is verified whether the name of an actor is unique. The logical actors further contain checks that e.g., the initial stock levels and the reorder points have positive values.

Simulation Building Block Guideline 17: parameters in a user interface of a simulation building block should have default values whenever possible.

Simulation studies in the domain of supply chains are performed for a wide range of systems. It is impossible to define valid and logical default values. In a supply chain for computer parts the order sizes and delivery times are completely different than in a supply chain for airplane engines. Nevertheless, the user interfaces are equipped with default values. The default values in the simulation building blocks are not to show possible valid values, but to show that values should be filled in and to avoid that the simulation model will return errors caused by faulty data entry.

Simulation Building Block Guideline 18: The user interface of a simulation building block should provide support for the model developer.

The description of the parameter in the user interfaces of the building blocks and building block elements is made as self-explanatory as possible. Additional explanations of each individual parameter are described in the user documentation of the domain specific extension on top of Arena in case the self-explanatory parameter description is insufficient. The users of the eM-Plant simulation environment have to do without, as no time has been spent on additional user documentation.

Simulation Building Block Guideline 19: The user interface of a simulation building block can be used by model developers to select building block elements and set their parameters.

The selection of the building block elements is differently handled in each simulation environment. Figure 6.6 shows for Arena how a model developer instantiates the simulation building block into the simulation model, in this case a logical manufacturer. One-by-one the required building block elements have to be added in the Arena case.

The model developer in the eM-Plant environment can use the drop down list in the user interface (Figure 6.3) or manually replace building block elements within the building block (Figure 6.4) by removing the existing building block element and instantiating a building block element from the library of the same family.

Simulation Building Block Guideline 20: a simulation building block has a defined interface that receives triggers, requests, entities, or events from other simulation building blocks in the simulation model and redistributes these internally.

Figure 6.5 and Figure 6.8 demonstrate how this is handled in eM-Plant and Arena via mail boxes.

Simulation Building Block Guideline 21: the interface of a simulation building block contains evaluations of the state of the trigger and the building block to determine whether the building block can handle the trigger.

Figure 6.8 shows that an error will be generated if a simulation building block or a building block element receives an incoming entity that triggers functionality that is not applicable for the simulation building block. Similar checks are also performed if triggers or messages are received at moments that the simulation building block or building block element is not expecting it given the state.

Some examples are:

- Receive a quote when no request for quotation is send
- Receive a quote from a logical actor to whom no request for quotation is send
- Receive a notification from a physical actor that an order is received when no order was placed
- Receive a bill when no order was placed

In both simulation environments these checks are implemented in similar ways.

Simulation Building Block Guideline 22: a simulation building block contains pictures, numbers and other elements to support visualization of the state and key performance indicators during simulation run.

The supply chain building blocks implemented in Arena have been provided with animation details to visualize the state of the logical actor. Figure 6.12 shows for example information related to the duration of certain activities, the financial situation of the actor and stock levels. More information is available regarding the state of the logical actor, but that would overload the visualization. Therefore this information is all exported to a dedicated MS Access database that collects and manages all events and statistics in the system.

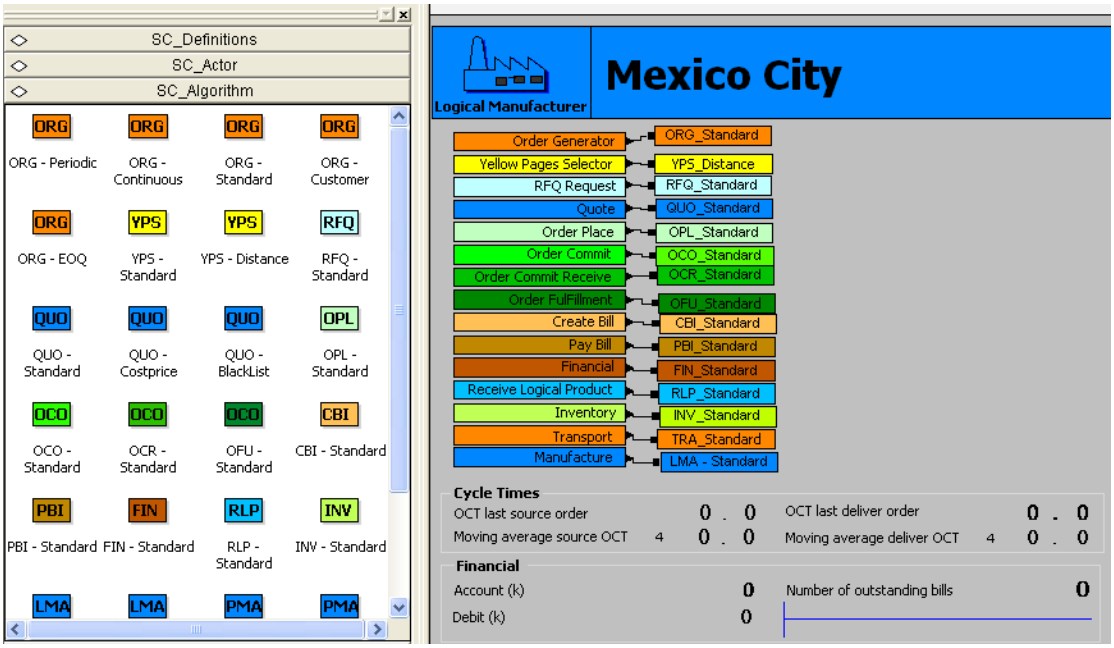


Figure 6.12: Visualization items of logical actor

6.6 Simulation studies performed

6.6.1 Fictive computer supply chain

Corver (2001) demonstrated the applicability of the domain specific extension using a fictive computer supply chain with world-wide suppliers and

manufacturers providing products to customers in the USA and Australia. This same case study has been used by Van der Hee (2002) as an example model in Arena. The product manufactured in the fictive supply chain is a PC. The manufacturing process is mainly an assembly activity from a keyboard, monitor, casing, mouse and speakers, which can be provided by one or more suppliers. The manufacturer assembles the PCs for USA or Australia according to the so-called Bill of Materials. In the example, there are two different manufacturers. Both manufacturers can provide PCs for the USA, but only the manufacturer in Taipei can manufacture PCs for Australia. In the concept there are three distribution centers. The distribution centers in Salt Lake City and Frankfort serve the US market. The third distribution center, in Alice Springs, serves the Australian market. In the concept, customers place orders only at one of the retailers. Figure 6.13 shows the actors that participate in the fictive computer supply chain and the potential flows of products.

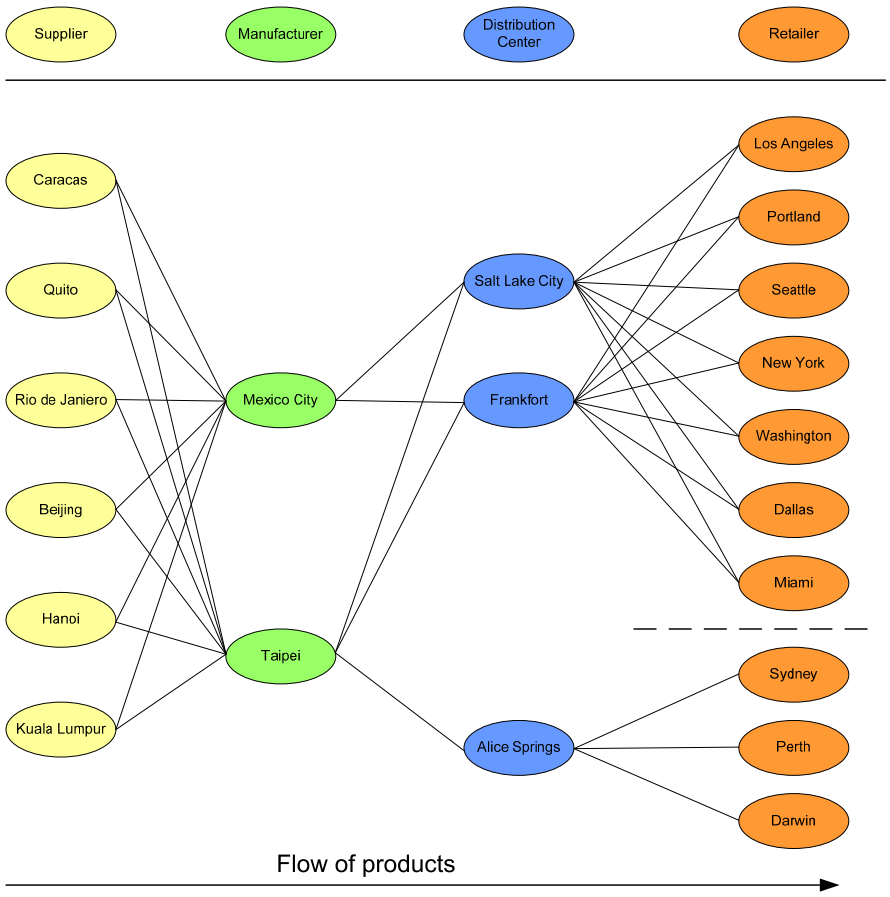


Figure 6.13: Layout of fictive computer supply chain (Corver, 2001)

The suppliers of raw materials (left actors in Figure 6.13) provide different components that are used by the manufacturers in Mexico City and Taipei (second set of actors in Figure 6.13). The different suppliers provide different materials. The materials that suppliers need for their production process is

kept outside the scope of this fictive supply chain. The manufacturers produce and assemble the physical products from the suppliers and their products are transported to the distribution centers (set one from the right). The distribution centers send the physical computers that they receive to retailers and retailers finally get purchase orders from customers in their neighborhood. In this supply chain it is not possible to skip actors, e.g. there is no direct delivery from manufacturer to retailers possible.

All actors can have product in stock and will place requests for quotations and finally orders to providers down the supply chain if they run out of stock. If an actor cannot satisfy an order, then a backorder will be placed and as soon as the stock is replaced the pending backorders will be satisfied.

Figure 6.14 shows the building blocks of the domain specific extension of eM-Plant with at the background the building blocks instantiated in a simulation model and situated across the USA with the manufacturer, distribution center and retailer.

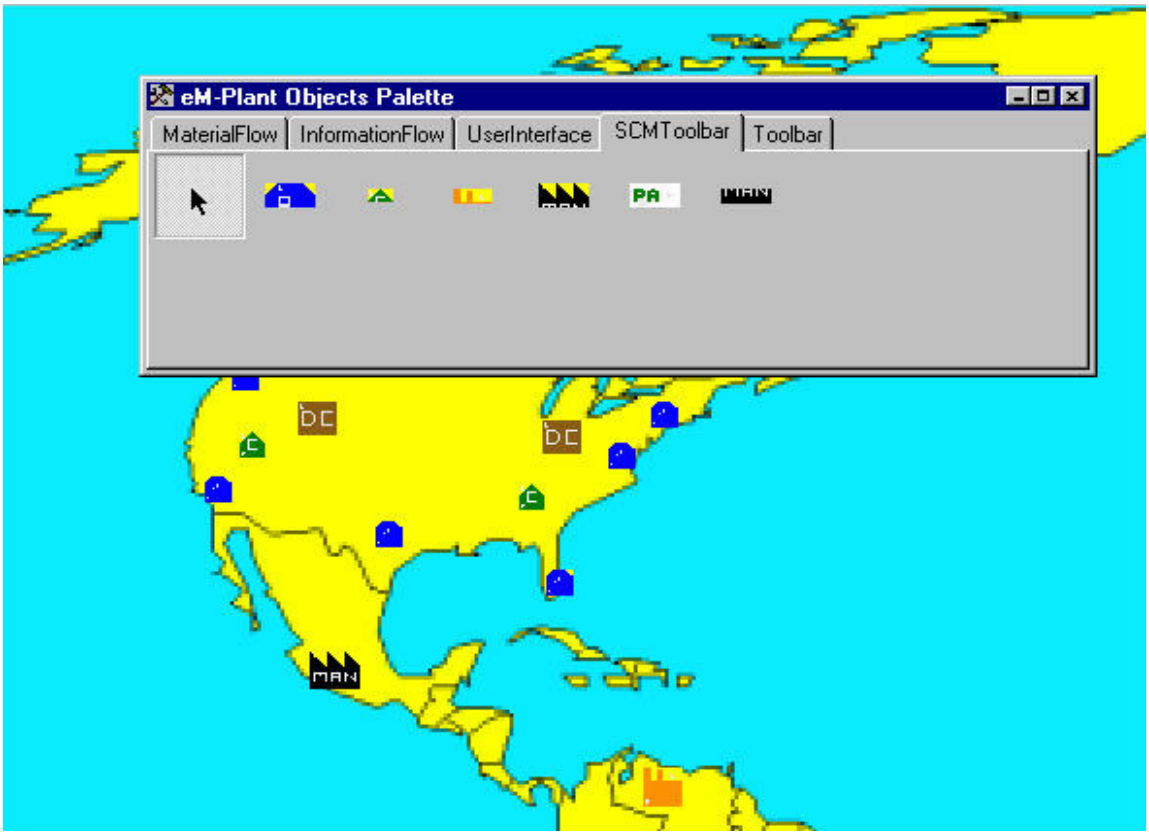


Figure 6.14: Supply chain actors placed on the world map (Corver, 2001, p50)

The aim of the simulation model was to prove the concept of handling logical and physical actors with their different sub-processes to represent particular behavior. The instantiation of the building blocks according to the case description and behavior of the logical actors to the limited scope proved suitable.

Corver (2001) demonstrated the applicability via different configurations and roles of the actors in the simulation model. One example was a simulation experiment in which the manufacturer in Mexico City was closed for one complete year. During this year Taipei has to supply to the distribution center in Australia and has to satisfy the full requirements of the distribution centers in the USA. Taipei consequently has to deal with backorders, because the production facilities are running full time.

6.6.2 Simulation models for teaching at R.H. Smith Business School

The domain specific extension for supply chains in Arena was used to develop a set of demonstration models and teaching cases. The demonstration models were used to show the effect of decisions and policies. For these teaching examples, the same fictive computer supply chain has been modeled as described in Figure 6.13.

The teaching cases were not just demonstration models, but they encouraged the students to perform an analysis and experiment with the basic situation. Three different teaching cases asked the students to come up with the best possible policy and settings. The first teaching case regarded a simple optimization and parameter setting of an already existing supply chain. The second teaching case regarded a certain change in demand and required the students to lessen the bullwhip effect in the chain, and the third teaching case asked the students to handle uncertainty in demand with either a distribution center or direct ordering at the manufacturer. A part of the simplest teaching case is shown in Figure 6.15.

Assignment 1: Connect Inc.

Joe is a manager at Connect Inc, a business telephone manufacturer. He's looking into ways to improve his fill rate's on customer orders (or reducing stockouts), while at the same time lowering inventory levels.

He has asked you, the business analyst, to make an analysis of his supply chain and present a number of solutions to his problems. Where possible, he wants the improvements clarified with numbers.

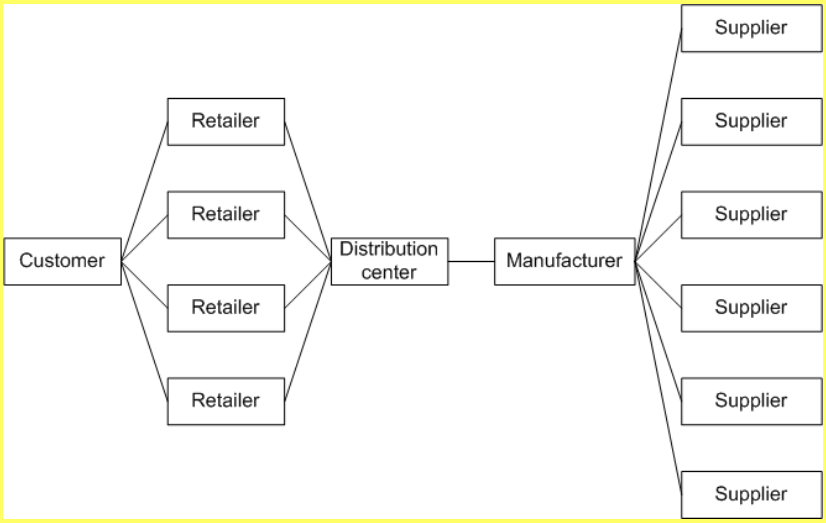


Figure 6.15: Fragment of a teaching case (Van der Hee, 2001a, p14)

The teaching cases were based on descriptions of supply chain cases provided by the teaching staff of R.H. Smith Business School. The teaching staff had expectations for the simulation model behavior for each of the teaching cases, mainly based on theoretical knowledge. The input data and complexity of the case has been adjusted until the desired behavior was represented by the simulation model in different configurations and with different selections of decision algorithms for actors in the supply chain.

The teaching cases provided directions for the students to the type of experiments they could do, but the teaching cases were open ended by design and provided students with opportunity to experiment themselves.

The demonstration models and the teaching cases have been used in a course for the MBA-program of the R.H. Smith Business School, University of Maryland, USA 20 groups of 2 to 3 students analyzed the outcome of the simulation models and improved the models by adjusting the parameter settings for ordering and storage, as well as by changing policies. The result and learning effect was satisfactory according to the teachers of the supply chain course (Van der Hee, 2001; Verbraeck, 2004).

6.6.3 Simulation study for Department of Defense of the turbine engines of the F101

The Department of Defense (USA) has a complex supply chain to provide maintenance parts for their airplanes and helicopters that are in active service worldwide. The parts of an airplane are stored at dozens of places worldwide at intermediary distribution centers, and in some cases an order is placed to the manufacturer who is also organized globally. The supply chain experts of the Department of Defense thought that a reduction of their storage cost and the number of out-of-stock events should be possible, if they would have better knowledge of their current requirements and the possibility of transferring maintenance parts through their network (Jacobs, 2005).

A proof of concept project has been carried out for simulating the supply chain of the Low-Pressure Turbine for F101 engines focusing on 25 parts of the engine. One of the requirements of the study was that the information of the supply chain should be spread over the internet and only limited information can be provided and responded to, due to security and safety. The information of the current situation should be reproduced and information from different sources should be collected and reproduced in a simulation model of the supply chain. Based on this information a simulation model should run to provide insight when a certain part could be available and how many parts should be stored for the coming period of time at the different locations worldwide.

The simulation model of the turbine engines of the F101 is developed as a Java program based on the DSOL simulation engine that will instantiate a world map and use information from the specific databases to instantiate actors on geographical locations at the world map. This world map and certain parts of the information available at actors that the user of the simulation

model is permitted to view can be accessed via a web browser or applet as shown in Figure 6.16.

The order information that is used in the simulation model is also retrieved from the available data sources and result in actual transaction and information exchange between the actors, for example the animated airplanes moving from the manufacturer to a base in mid USA (Figure 6.16).

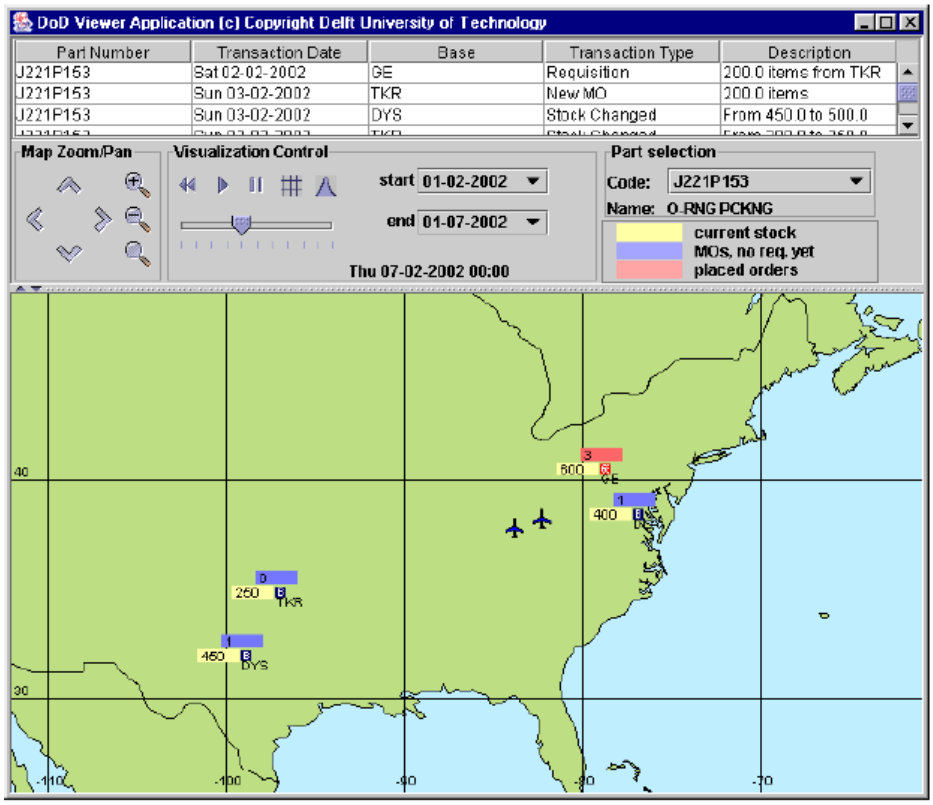


Figure 6.16: Viewer of simulation model F101 supply chain (Verbraeck, 2004)

The system was evaluated by representing the system in two phases, first the modeling of the current activities in the supply chain based on historical information. Secondly it was verified whether the simulation model represents similar behavior as visible in historic data files. Evaluation by experts of the Department of Defense supply chains showed that the future behavior was in line with the expectations based on the historical data files.

The involved actors of the Department of Defense were impressed by the possibilities of simulation and identified future extensions of the use of real-time management of supply chains with simulation models. However, the complexity of gathering the data in this minor proof-of-concept project and the required exchange between different actors turned out to be a time consuming factor (Verbraeck, 2004).

6.7 Observations during simulation studies

The main observation from the simulation studies performed with the different implemented domain specific extensions is that the design of building blocks and building block elements is not hindered by the generic simulation environment in which the building blocks will be implemented. In all three generic simulation environments the developers succeeded of getting the building block and the basic building block elements running according to the Simulation Building Block Guidelines identified in chapter 5.

6.7.1 Observations regarding design approach and implementation

A positive observation was the capability of designing a consistent domain specific extension, starting from a problem domain description. A negative observation was that the initial design of the domain specific extension was not easy to apply for real-time simulations based on a historic starting point, which was necessary for the simulation study for the Department of Defense. This type of use is not very common in the domain of discrete event simulation and the standard simulation environments are not prepared to provide these kinds of services. Nevertheless, this type of experiment should have been included in the definition of the scope and the problem domain of this domain specific extension. Including the experiment might have provided some additional requirements for the design of the building blocks. Another negative technical observation was that Arena as a generic simulation environment was not able to handle more complex data types needed to simulate supply chains (Van der Hee, 2002).

6.7.2 Observations regarding additional tools

No observations are available of additional tools for the domain specific extensions in eM-Plant and DSOL as no additional tools have been built. The users of these domain specific extensions also had no complaints about the absence of additional tools. The comments for the additional tools developed for the domain specific extension developed with Arena are both positive and negative.

Positive is that the MS Access database collected the needed data and enabled the students to make comparisons, but it could have been even better if the MS Access database would have contained more reporting instruments that would have helped the students of the teaching cases to focus on the important data and more easily derive conclusions.

Positive about the VB-Program that supported the model development was that it helped to quickly instantiate a new actor with all of its building block elements. Negative about this tool was that it only worked for the initial instantiation. Once the actor building block with its building block element was in the simulation model, the VB-Program could not help with adjusting. Furthermore, the development of the VB-program started from the observation

that creating the building blocks and the building block element structure in Arena was time consuming.

6.7.3 Observations regarding provided support

Positive about the support was the wide availability of support material for model developers and model users for the implementation in the Arena environment. Positive was also the direct availability of live experts for the domain specific extensions for eM-Plant and DSOL, although that kind of support would not be available as soon as the domain specific extensions would have been handed over to other model developers.

A possible improvement could be the limited integration of the support material with the building blocks. Now all the documentation was provided on a separate CD that model developers could access, while the information could be integrated with the Arena simulation building blocks using the development of a help file.

6.7.4 Observations regarding applying building block concepts and guidelines

The observations for the capabilities of the domain specific extension are structured via the characteristics of a building block as defined by Verbraeck et al (2002) in Table 6.2: Characteristics of building block in Supply Chain case study.

Table 6.2: Characteristics of building block in Supply Chain case study

Self-Contained	Positive	data locally stored & use of building block elements and alternatives for functions.
Interoperable	Positive	families of building blocks enable the exchange of building block elements.
	To be improved	division between infrastructure and control
Reusable	Positive	design of building blocks applied in three different generic simulation environments, and actor structure applied to all actors in the supply chain.
Replaceable	Positive	extended sets of replaceable building block elements.
Encapsulating its internal structure	Positive	eM-Plant building blocks were closed, yet an expert could open and improve them.
	Negative	Arena building block elements completely closed and no capability of extending with generic model constructs.
Providing useful services	Positive	building block element for all defined functionalities of actor in supply chain.
Precisely defined interfaces	Positive	structure of building block and building block element enabled replacing of building block elements without risk.

It turned out in all three domain specific extensions that the developers are capable of extending the set of simulation building blocks and building block elements with additional predefined functionalities. Unfortunately, in this case study we did not have the opportunity to apply the building blocks of the domain specific extension to a real-life simulation study rather than a teaching case or a proof of concept. Therefore, this case study does not demonstrate that the building blocks or the building block elements have the required flexibility to be extended beyond the original designs of Corver (2001). On the other hand, there is no reason to assume that the building blocks and the building block elements cannot handle this flexibility, especially when observing the extended set of close-to-real-life supply chain concepts that are represented in the teaching cases by Van der Hee (2001).

6.8 Overview observations case study Supply Chain

6.8.1 Observed of benefits

The simulation studies that were performed confirmed the benefits that were noted in chapter 2 and 3. Overall, the simulation studies were performed correctly. Table 3.6 provides a similar summarizing overview as presented in chapter 3 tables 3.6 and 3.7 regarding only the benefits.

Overall all benefits have been achieved, but the benefits have not been shown in all simulation studies, because the development has been different depending on the generic simulation environment. For example, the benefit ‘semi-automatic reporting of performance indicators’ is only achieved by the Arena implementation thanks to the MS Access database implemented as an additional tool (Van der Hee, 2002).

Table 6.3: Summary of benefits observed in case study Supply Chain

Process step **Expected benefits as mentioned in chapter 2 and 3**	**Observation supply chains**
Activity 1: Problem description & define conceptual model	
Benefit 1.1: conceptualize system elements with model constructs in mind	Yes
Activity 2: Select model constructs	
Benefit 2.1: no translation between system elements and model constructs	Yes
Benefit 2.2: compose model constructs from developed domain specific model constructs to represent system elements	Yes
Benefit 2.3: easy selection of model construct thanks to structure of domain specific extension	Yes
Activity 3: Data collection	
Benefit 3.1: collection of predefined input data	Yes
Activity 4: Instantiate simulation model for original system	
Benefit 4.1: less model constructs used	Yes
Benefit 4.2: model development faster and easier	Yes
Benefit 4.3: model development by simulation novices	Yes
Activity 5: Verify and validate simulation model for original system	
Benefit 5.1: no more detailed testing	Yes
Benefit 5.2: easily gathering validation data	Yes
Benefit 5.3: structured and standardized performance indicators	Yes
Benefit 5.4: semi-automatic reporting of performance indicators	Yes
Benefit 5.5: observe animation at different levels of the composition: high level and at individual model construct	Yes
Activity 6: Analyze output of simulation model	
Benefit 6.1: structured and standardized performance indicators	Yes
Benefit 6.2: semi-automatic reporting of performance indicators	Yes
Activity 7: Define solution for analyzed outcome	
Benefit 7.1: model developers are triggered to find new solutions by parameters	Yes
Activity 8: Instantiate simulation model for identified solution	
Benefit 8.1: easy adjustment of model thanks to user interfaces of model constructs	Yes
Benefit 8.2: easy adjustment of model thanks to replacement of model constructs	Yes
Benefit 8.3: easy visualization thanks to incorporation of visualization in model constructs	Yes
Benefit 8.4: composition of new model constructs enabled new solutions to be evaluated	Yes

6.8.2 Observed of risks in Supply Chain

The simulation studies that were performed in the different simulation studies also provided confirmation about the capability of the domain specific extension to mitigate the risks that have been identified in chapter 2 and 3. Thanks to the Simulation Building Block Guidelines and the design approach of chapter 5 we seemed to have mitigated most of the risks described in chapters 2 and 3.

Table 6.4 provides a similar summarizing overview as presented in chapter 3, tables 3.6 and 3.7 regarding only the risks. Potential risks that we have mitigated during the execution of the case study ("No" in the table) did not occur and the potential risk has most probably been avoided by the way the domain specific extension was designed, structured and used.

Table 6.4: Summary of risks observed in case study

Process step **Potential risks as mentioned in chapter 2 and 3**	**Observation supply chains**
Activity 1: Problem description & define conceptual model	
Risk 1.1: scope of model developer is limited by model constructs	No
Activity 2: Select model constructs	
Risk 2.1: lack of trust results in no motivation to use domain specific extension	No
Risk 2.2: lack of insight in model constructs results in ignore domain specific extension	No
Risk 2.3: use of model constructs that are not suited for representation of system elements	No
Risk 2.4: system elements can not be represented by model constructs	No
Risk 2.5: compose model constructs from developed domain specific model constructs only applied for infrastructure system elements	No
Risk 2.6: model developers can adjust internal logic of model constructs	No
Activity 3: Data collection	
No risks defined in chapter 2 or 3	
Activity 4: Instantiate simulation model for original system	
Risk 4.1: model developers do not understand model construct	No
Risk 4.2: model developers do not know how to parameterize model construct	No
Risk 4.3: difficult to compose simulation model, because model constructs are not available	No
Risk 4.4: difficult to compose simulation model by person other than developer(s) domain specific extension	No

Activity 5: Verify and validate simulation model for original system	
Risk 5.1: mistakes of model developer are hard to overcome	No
Risk 5.2: model developers know something is wrong, but cannot identify what to do about it	No
Activity 6: Analyze outcome of simulation model	
Risk 6.1: model constructs do not provide performance indicators problem owner desired	No
Activity 7: Define solution for analyzed outcome	
Risk 7.1: model developers are triggered to find new solutions by parameters	No
Risk 7.2: model developers are limited by parameters and model constructs	Partly
Activity 8: Instantiate simulation model for identified solution	
Risk 8.1: solution is identified that cannot be represented by model constructs	No
Risk 8.2: adjustments of model constructs required to represent solution are time consuming	No
Risk 8.3: replacement of model constructs causes errors in model constructs that were linked or connected.	Partly

The risks encountered by the domain specific extensions for supply chains do not apply to all implementations. The risk ‘system elements cannot be represented by model constructs’ and ‘model developers are limited by parameters and model constructs’ existed in eM-Plant and the DSOL implementation as a result of the scoping. Not all designed building blocks have been implemented and thus not all systems could be represented (Corver, 2001; Tewoldeberhan, 2005). The risk ‘model developers can adjust internal logic of model constructs’ only applied for the eM-Plant implementation, whereby sufficient warnings in the user interface were included to make sure a model developer was aware of the risks of his/her actions (Corver, 2001).

The risk ‘replacement of model constructs causes errors in model constructs that were linked or connected’ appeared in supply chain teaching cases with Arena when the model developers included a mixture of building block elements for the concept ‘make-to-order’ and ‘make-to-stock’. Several building block elements needed to be replaced to change from one to another concept.

7 Application to container terminals

7.1 Why develop a domain specific extension?

The planning and design of infrastructures, such as railroads, business parks and utility networks, is a complex task due to a large number of interrelated design parameters and mutually dependent public and private stakeholders. Possible stakeholders such as local, regional and national public authorities, infrastructure operators, (potential) customers, logistical, transportation and shipping companies, residents and environmental associations, constitute an inter-organizational network. (De Bruijn and Ten Heuvelhof, 2000). The actors are mutually dependent, but will most likely have different perceptions, interests, values and objectives. As a consequence, the optimization of technical, economic and logistical values will strongly be inhibited by conflicting interests, political and external boundaries and strategic stakeholder behavior.

Visualization and simulation can contribute to the initial design phase of complex infrastructures. Visualizations of a system, such as sketches or layouts, and simulation models, expressing a system's dynamic characteristics, communicate the complexity of the system, show the consequences of options and place a design in its future environment. Important questions can be raised however, on how collaborative visualization and simulation can be embedded in an inter-organizational network and a multi-actor negotiation process. Can visualization and simulation contribute to the quality and progress of negotiation for instance by facilitating the development of mutual understanding or a shared vision? In that case, what are the specific requirements and guidelines for using visualization-simulation tools in an inter-organizational context? What interactive procedures, programs and ground rules may guide such a collaborative vision development? (Mayer et al, 2004)

To answer the aforementioned research question a gaming-simulation (Duke, 1980) was developed using the planning and design of a fictitious but realistic inland container terminal as the topic of the game. In the gaming-simulation 'Containers Adrift' the participants play the role of stakeholders (Mayet et al, 2004). They explore and negotiate a container terminal design whereby all stakeholders are in principle in favor of the container terminal, but only if certain conditions regarding either economic scale, noise production, sight disruption or passing trucks are met. The stakeholders are supported during the game by a visualization-simulation tool that provides them the opportunity to evaluate different designs of the container terminal and observe the results for key parameters like operational profit, truck movements, initial investment, noise and CO_2 production. The basis of the visualization-simulation tool is a domain specific extension consisting of simulation building

blocks that are used to create and automatically execute a simulation model based on the design and choices of the stakeholders regarding the container terminal.

The aim of the visualization-simulation tool was to enable the participants to evaluate effects of design characteristics of the important elements for a terminal design such as its location, the quays, roads, ships, trucks, cranes and containers (Du, 2002a; Bockstael et al, 2003; Mayer et al, 2004). The visualization-simulation tool should provide quantified indicators for the performance, which were used by the stakeholders in the negotiation during the game. Therefore the complete cycle of the design and parameterization, model development, model run and evaluating the results of the simulation model should be within time boundaries of 15 minutes, otherwise the game-participants would not have sufficient patience to use the results in their negotiations.

7.2 Initial team to develop domain specific extension

The domain expertise for the container terminal was delivered by the game designers who defined the scope of the game for the design of the container terminal. They tried to be realistic, but also aware of the complexity to design a game with ten different stakeholders who all should have the opportunity to express their position (Van Kempen et al, 2002). The team of game designers and the game development team were experts from the TU Delft (Bockstael et al, 2003; Mayer et al, 2004).

7.3 Specification of domain specific extension

7.3.1 Scope

The scope of the visualization-simulation tool, and thus of the domain specific extension was defined by the background of the game situation: A new inland container terminal where containers where stored and picked up and delivered by trucks or vessels (Van Kempen et al, 2002). Inside the container terminal the transport was handled by special vehicles that were dedicated for the terminal and cranes that lifted the containers in and out of the vessels. The experiments that should be performed were mainly to support the game participants in making decisions on the capability of the container terminal. The decisions provided in the game design that had to be supported by the visualization-simulation tool were the following:

- Size of the storage location (square meters and height)
- Number and type of customers (resulting in truck movements and storage occupancy)
- Vessel size and pattern
- Number of dedicated container terminal vehicles
- Number of cranes for loading/unloading vessels

The participants in the game had some additional decisions that they needed to make, but these were not supported by the visualization-simulation tool. The main decision was the location of the terminal, close to the city or close to a nature park. Even though the selection of the location did not affect the outcome of the visualization-simulation tool, the decision was mainly based on the outcome of the tool concerning performance indicators as noise and CO_2 production (Bockstael et al, 2003).

Other performance indicators that the participants used (or decided to ignore depending on their role in the game) in their negotiations were: initial investment, operational profit and the number of jobs (Bockstael et al, 2003). These performance indicators were helpful for the negotiation, but to improve the design of the terminal (and thus enlarge the profit or reduce the investment) the participants required insights in turnaround times of vessels and utilization of storage space, cranes and trucks.

The game design and the limitation of the duration of the game also put some types of experiments clearly out of scope of the visualization-simulation tool. Examples are the ability to change to process of handling a container in the container terminal, selection of storage space in the terminal and priority settings in vessel and customer handling. Whenever participants came with the question whether the visualization-simulation tool could provide any insights resulting from adjusting these mechanisms, the answer was that they should identify this as a possible gain for the design and steer the negotiation towards future research.

7.3.2 Problem domain

An inland container terminal consists of objects that are used for the transport of containers. The containers are delivered and picked up using trucks or vessels and moved by cranes and forklifts at the terminal area. The additional objects for an inland container terminal are a quay, storage space and roads where forklifts move around. The processes that occur in an inland container terminal and the decisions that are made in the terminal are described in special building blocks and more detailed building block elements for dedicated processes (Du, 2002a).

The advanced processes in a container terminal are evaluations such as: where is a container terminal stored, which containers are leaving in a vessel and which sequence is used in loading the vessel. These evaluations are important in a realistic and technical representation of a container terminal, but the simulation models as part of a game for initial design of a container terminal do not require advanced algorithms. The order and sequence can be determined by following a first in-first out strategy, resulting in relatively simple processes.

7.3.3 Building blocks

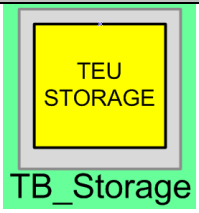
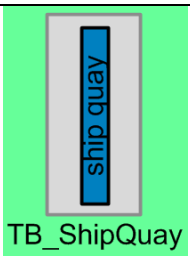
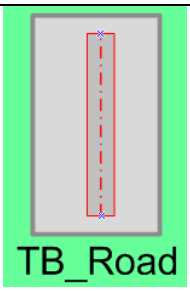
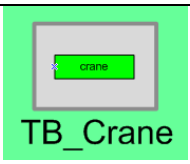
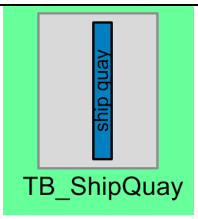
The building blocks that have been defined in the domain specific extension are a larger set than what the designer of the container terminal encounters (Du, 2002a). Table 7.1 shows the building blocks that form the

physical container terminal that the model developer instantiates in the VISIO drawing tool. These building blocks contain the physical representation and the mechanisms to claim and release the physical availability of another object, i.e. the storage space is occupied by containers and the ship quay is occupied by one or more vessels.

In addition to the physical elements that are drawn at a fixed position of the terminal layout are the movable entities, i.e. the trucks, vessels and internal vehicles. These are designed as a building block together with their control mechanism that determines what activities the movable entity should perform.

The last set of building blocks are at the boundary of the system, the companies that generate containers to be transported and the harbor, the location where vessels arrive or depart from.

Table 7.1: Building blocks in VISIO drawing with their description

Building block	Picture in VISIO	Description
Storage		Location where containers are stored.
ShipQuay		Location where cranes are placed and vessels board until they are loaded / unloaded by the available cranes.
Road		Infrastructure used by the internal vehicles to transport containers from the storage to the quay or the truck parking spot.
Crane		Equipment to load or unload containers from the internal vehicles to the vessels.
Parkingspot		Location where containers are loaded or unloaded from trucks by the internal vehicles.

7.3.4 Building block elements

The domain specific extension for container terminals contains no variance in building block elements within a building block. Each building block has only one way of performing its functionality. However, certain building block elements are applied in different building blocks of the domain specific extension.

The physical building blocks shown in Table 7.1 all contain a building block element to claim their capacity, to release their capacity and to gather statistics to be reported to the dedicated Excel report (see page 204).

Further building block elements that are defined in this domain specific extension are mainly used to structure the functionality of the building blocks (Du, 2002a). For example, the building block 'Company' consists of four building block elements: Create container to be picked up by truck; create container to be picked up in harbor; handle container arriving at company by truck; handle container arriving at harbor by vessel. The structuring of the building block along the lines of this structure helps the development, interfacing and debugging, but does not have any further added value to support future extension and development of the domain specific extension. The main reason for this simplification was the clear starting point that no variations for building block elements were to be used by the game participants.

7.4 Implementation

The development of this domain specific extension focused on the requirement to be able to perform the complete simulation and evaluation process with the visualization-simulation tool within 15 minutes by game participants. The implementation therefore focuses mainly on the capability of automating the model development, and to run and experiment based on a solid set of building blocks that represent the behavior of system elements of a container terminal (Valentin et al, 2002).

7.4.1 Extension of simulation environment

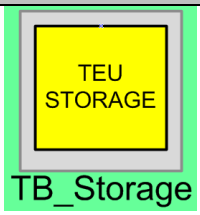
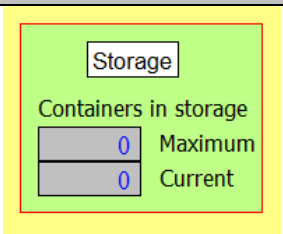
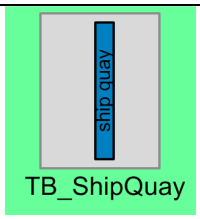
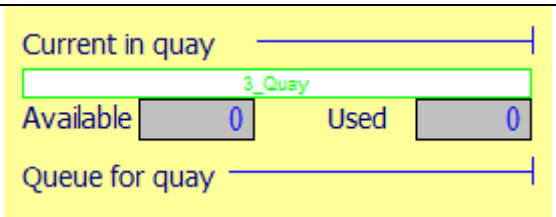
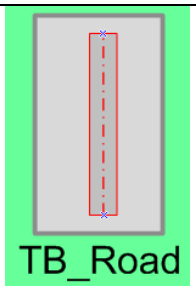
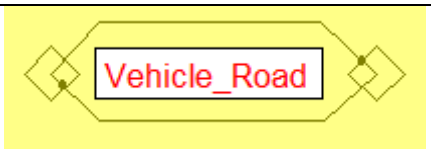
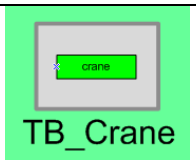
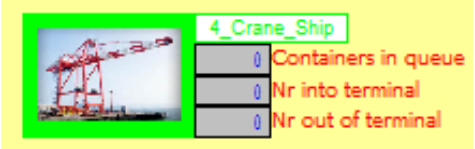
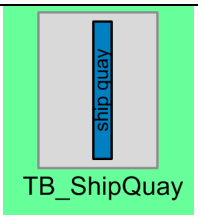
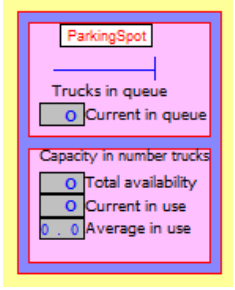
The selected generic simulation environment was Arena, with as the main reason the ability to automate model development via Visual Basic coding from an external application. An additional reason was the ability to develop simulation building blocks that were completely closed, to avoid that game participants would be able to alter the functions of the system and thus alter the basis of the game negotiations.

The processes in the simulation building blocks are triggered by a container entity created by the Company building block. This entity further follows the fixed modeled steps of transport via truck to the terminal, storage in the terminal, move to a vessel, transport by vessel to harbor or the other way around. The container entity is active at all times and executes the functions of claiming and releasing the physical resources. Inside the Arena

building blocks no ‘supervisor’ functionalities are developed and the logic in the claim building block elements is all ‘first come, first serve’.

The container entity is routed between the different physical resources via ‘mail box’ routing identified with specific codes. For example, code 11.4 is for claiming the container crane to leave the vessel, code 11.5 for the unload process by the container crane and code 11.6 for releasing the container crane (Valentin et al, 2002).

Table 7.2: Building blocks in VISIO and ARENA with status representation

Building block	Picture in VISIO	Representation in simulation model
Storage		
ShipQuay		
Road		
Crane		
Parkingspot		

The building blocks implemented in Arena automatically contain a user interface for the parameter setting and a visualization of their state. The user interface and the location of the visualization of the building block are filled via the automatic model generation tool (see next section) and the model developer does not need to pay attention to set the parameters or review the status. Even though there is no need for it, the user interface and the visualization are provided with the terminology used within the game to enhance the ability to understand the model if game participants are interested. Table 7.2 contains the visual representation of the physical building blocks earlier on described in Table 7.1. The representation in the simulation model shows the state of the physical resource, for example with numbers to indicate the available storage capacity and with colors to show the state of the container crane.

7.4.2 Additional tools

The domain specific extension consists of building blocks, the ability to translate a VISIO drawing into a simulation model, and an Excel interface that contains all performance indicators of the Inland Container Terminal. The building blocks of the physical elements of the container terminal are implemented in the drawing tool VISIO and the simulation environment ARENA. Figure 7.1 shows the technical sequence in the domain specific extension. A drawing of the terminal is made in VISIO and additional data of customer information is added to a MS Access database. The drawing and data are integrated into an Arena simulation model. The simulation run is performed and results are published in an Excel spreadsheet. In the remainder of the section a brief description of the different elements is provided. The full design can be found in Valentin et al (2002).

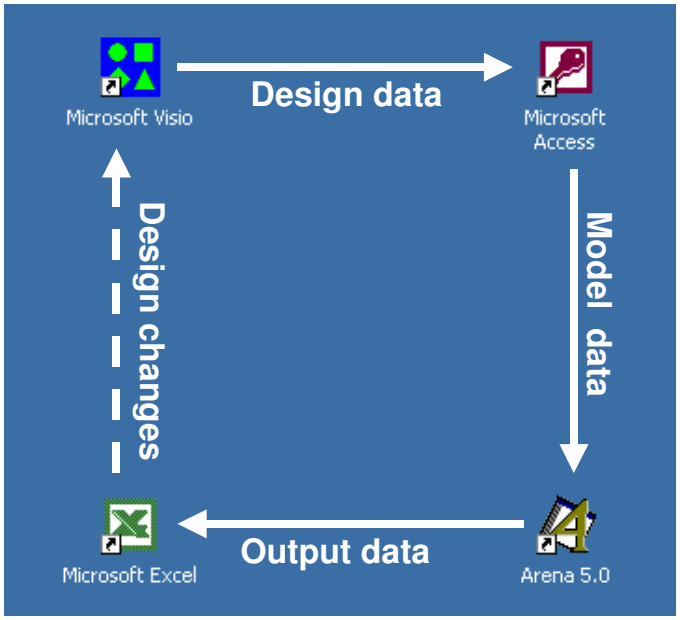


Figure 7.1: Cooperation between four tools

Model generation

The building blocks that represent infrastructure objects of a container terminal are represented in a VISIO drawing template and in an ARENA simulation template. The participants in the game make drawings using the VISIO objects. Their position and configuration are used to instantiate the represented building block in the simulation model. Table 7.2 shows the physical building blocks in both the drawing in VISIO and the simulation model in ARENA. The use of the drawing building blocks in VISIO can result in drawings of an inland container terminal such as shown in Figure 7.2.

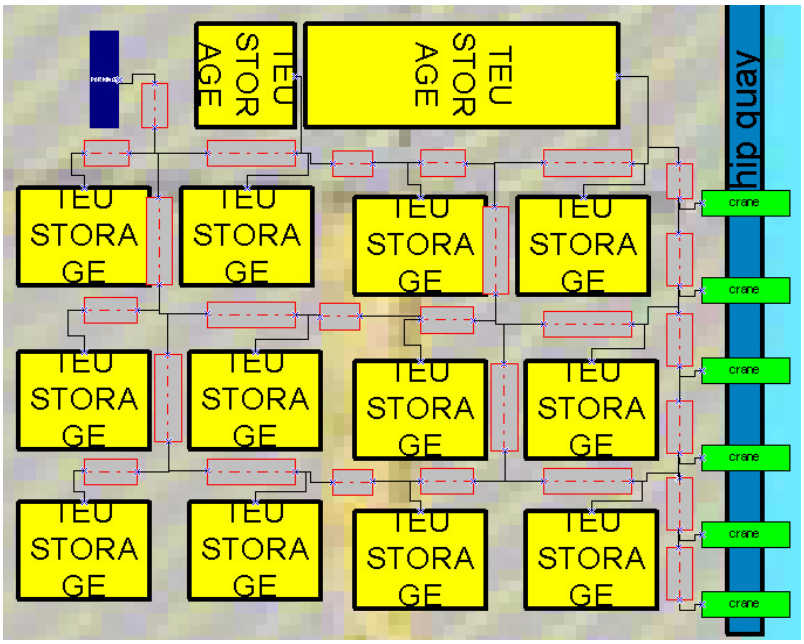


Figure 7.2: Example drawing of container terminal based on building blocks in VISIO

Within VISIO a Visual Basic (VB) program was developed that allowed the simulation model to be generated based on the drawing made. The VB-program analyzed all drawing elements in the VISIO drawing and registered which elements were based on the building blocks made available in VISIO to create the container terminal, i.e. the elements shown in Table 7.2. The registration included different attributes depending on the building block. For example, the ‘container storage’ registered the square meters and the height, the ‘ship quay’ registered the length and the ‘crane’ included the connections to the road network.

The VB-program followed by instantiating a new blank Arena model and instantiated in that model the building blocks for the physical system elements obtained from the VISIO drawing. This was followed by instantiating several building blocks for the transport, such as the trucks and the vessels. Finally the generation of the companies was done, and the simulation model was complete.

Model data entering

The VISIO drawing contains the configuration of the container terminal for the physical elements, but more data is required. For example, the number of trucks, the arrival times of vessels, and the companies that are supported by the container terminal. This information is all combined in an MS Access database in which the game participants can indicate their preferences. Figure 7.3 shows the selection the game participants can make between large or small vessels.

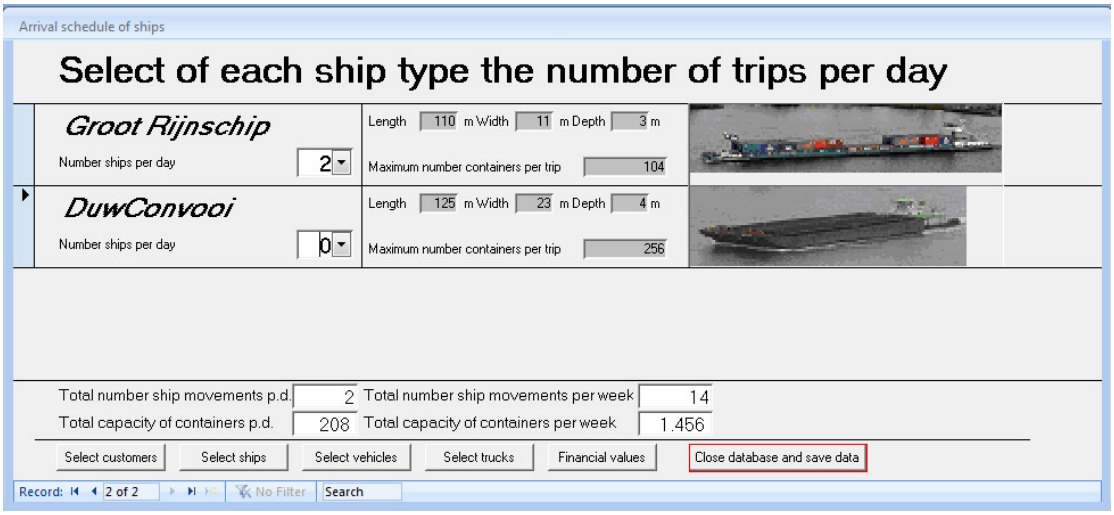


Figure 7.3: Access database for data entry, example vessel configuration

The information from the MS Access database is subsequently loaded into the instantiated simulation model via a VB-program similar to the program mentioned for the model generation. This VB-program copies all data of the MS Access database into the correct parameters fields of the respective simulation building blocks in the simulation model. The game participants do not need to perform any further manual activity to transfer the data into the simulation model.

Model execution

The game participants have two VB-programs for execution of the simulation model. One variant puts the Arena simulation model in the front of the screen and shows the visualization of the simulation building blocks filled with all the data from the MS Access database. While time proceeds the game participants can keep track of the state of the container elements and will see the trucks and internal vehicles drive around on the layout of the container terminal. The game participants will also see the visualization of the container cranes, the storage and the quay. Based on the visualization, they can decide whether their design seems feasible or whether it is a bad solution for the considered companies.

The other variant for model execution is a hidden execution. The VB-program executes the simulation model in the background and generates at the end of the simulation run all information required for the reporting in separate text files. The game participants do not see any progress of running the simulation model, but after 1 to 3 minutes they receive a message that the data is ready to be imported in the Excel reporting tool for analysis.

Model outcome reporting

The building blocks in the Arena simulation model provide functionalities to represent behavior of an inland container terminal and collect data for various performance indicators. The participants of this game need to evaluate different performance indicators, which are based on economics, logistics and environmental issues to reach consensus on one or two designs for the container terminal. The wide variety in performance indicators is needed to support the different roles in the game. For example, environmentalists are mainly interested in the noise and emission levels, while future customers of the inland terminal checked the throughput times provided. Further topics like queue length and utilization are included to support the participants in making suggestions for improvement of the terminal. Figure 7.4 shows a part of the Excel interface, representing the performance indicator for investments that are required for the design of the inland container terminal at the designated position.

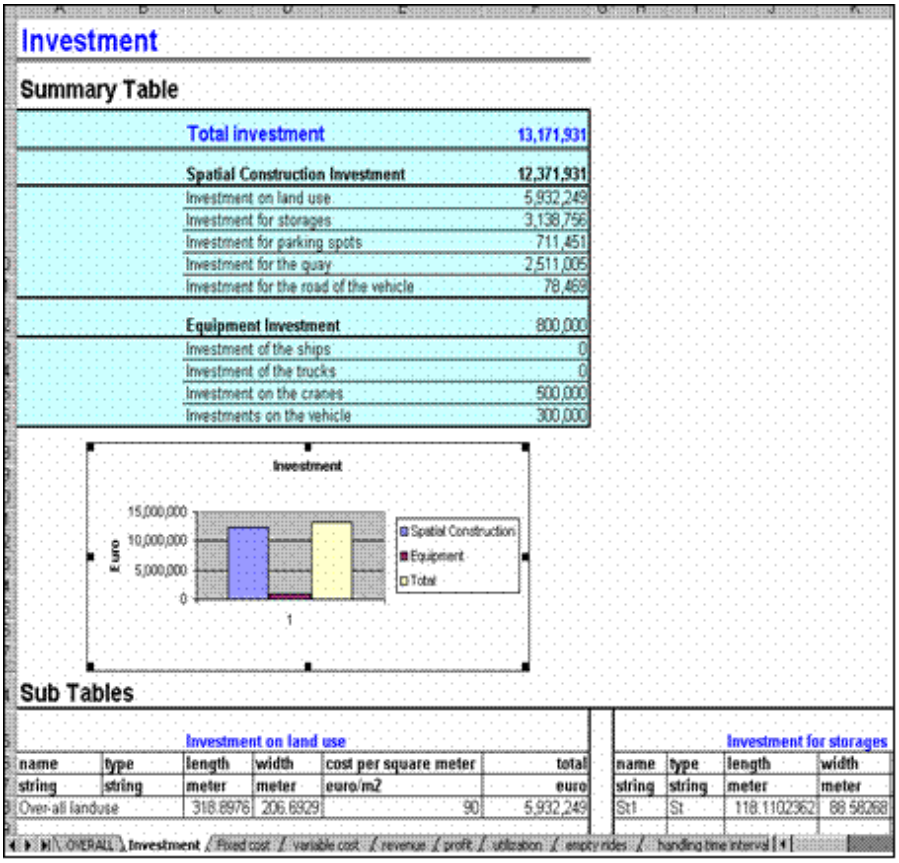


Figure 7.4: Excel interface showing performance indicators for investments

The data required for the reporting tool is automatically generated by the building blocks in the simulation model and stored in text files. A VB-program is triggered after the simulation model is executed to open a new instance of a template designed for the reporting. One by one the text files are read into the Excel sheet and their data is added to the appropriate sheets to calculate the performance indicators such as investment, cost, profit and utilization. The Excel template is also prepared with one overall A4 sheet showing all key parameters that are primarily used by the game participants during their negotiations.

7.4.3 Support to users

In the first hour of the game, the game participants receive an introduction in the game objectives, the different stakeholders, and the capabilities of the visualization-simulation tool. This explanation is accompanied by a quick demo of the complete process from an initial drawing in VISIO to the final report in Excel (Van Kempen et al, 2002).

Depending on the size of the group of game participants, between two and four workstations were set-up with a laptop, projector and paper support material where game participants together could design and evaluate container terminals. Each station contained all the necessary software and a couple of example drawings with simulation models (like Figure 7.2). In the first game each station was managed by a teaching assistant who knew the technical challenges and could guide the game participants around these problems. This teaching assistant also gave additional explanation on the spot about what the game participants saw on the screen during the simulation run.

The experiences gathered during the first game have been used to finalize a user manual and a ‘cheat’-sheet with the most common questions and explanations (Du, 2002b). The provided documentation and the attitude of the game participants to try themselves was a reason to reduce the on-site support and change to on-demand support. An expert was available during the following games to answer questions that popped-up regarding the use of the visualization-simulation tool and detailed questions (mainly about performance indicators in the Excel reporting).

7.5 Use of Simulation Building Block Guidelines

The Simulation Building Block Guidelines have been created mainly to allow future changes to the simulation building blocks and to make extensions to the domain specific extension. The domain specific extension as basis of the visualization-simulation tool that has been used by the game participants has not been changed since the first version of the game, with the exception of fixing a couple of minor bugs. The main upgrades have been carried out for the VB-programs to enhance the ability to automatically generate the simulation model with the correct data and produce a valid report for the

simulation run. Therefore, several Simulation Building Block Guidelines have not been followed for the development of this domain specific extension.

Guidelines related to self-contained building blocks

Simulation Building Block Guideline 1: data belonging to a building block should not be accessed by other building blocks directly, but only via defined interfaces.

The whole process is triggered by a container. The container accesses the information of the physical elements in the simulation model and is proceeding in the predetermined sequence, thus following the defined interfaces.

Simulation Building Block Guideline 2: a simulation building block consists of a core and building block elements to represent functions and services.

See the remark in section 7.3.4 regarding building block elements being hardly used, other than to structure the inner working of the building blocks.

Simulation Building Block Guideline 3: data belonging to a building block element can be accessed by other building blocks elements of that building block without using the interfaces of the simulation building block.

Building block elements are only used for structuring of the inner working of the building blocks in this domain specific extension. Therefore data did not belong to a specific building block element, but to the building block itself. For the communication and data exchange between building blocks the regular mail box system was used, see more in Simulation Building Block Guideline 9.

Guidelines related to interoperability of building blocks

Simulation Building Block Guideline 4: system elements that appear in different variants and processes in a system are represented by a family of building blocks and building block elements.

Some of the building blocks could be seen as ‘belonging to a family’, i.e. the physical building blocks used in VISIO and Arena (see Table 7.2), but no other families of building blocks have been defined. In line with the decision to keep the process simple and steady, no alternatives of building block elements are provided.

Simulation Building Block Guideline 5: building blocks are of different types, most common to have building blocks for infrastructure and for control.

The building blocks representing the physical infrastructure are listed, but they also include the functionality of claiming and releasing the resources, thus the control. In the type of experiments performed during the game, no advanced control was necessary, therefore it was decided in the design of the building blocks not to follow this guideline.

Simulation Building Block Guideline 6: complex control mechanisms should be represented using control building blocks linked together to represent a flow.

The level of abstraction in the container terminal was not sufficient to justify separate control building blocks that represent the process flow of a container. It was decided not to follow this guideline for the design of the building blocks.

Simulation Building Block Guideline 7: building blocks should be aware of each other's existence within a range of applicability.

In the simulation models interfaces were defined via the network of roads between the storages, parking spot and the container cranes. These were all defined in the VISIO drawing and automatically exported to the road-simulation building block. The internal vehicles used this information to transport a container, but no further flexible links were used.

Guidelines related to replaceable unit of building blocks

Simulation Building Block Guideline 8: extension of a domain specific extension can be achieved by introducing new building block elements for existing simulation building blocks.

Extending the amount of simulation building blocks within the domain specific extension was explicitly out of scope for the game. Sometimes game participants had additional questions for further research, which were not accommodated by the available building blocks or automatic model generation in the visualization-simulation tool. In these cases, the game participants received the reply that they should address the request in their negotiations as further research.

Simulation Building Block Guideline 9: simulation building blocks and building block elements of the same family follow the same interface requirements.

Each building block worked with the concept of 'mail box' to route the container entity and enable monitoring of its status. Each building block had its own code, because the building blocks could not replace each other's functions. For example, a storage location could not represent the functionality of the parking spot.

Guidelines related to encapsulating internal structure of building blocks

Simulation Building Block Guideline 10: simulation building blocks hide inner working.

One of the reasons to select Arena as the generic simulation environment is that model developers cannot interfere with the building blocks, except their parameterization. In this domain specific extension hiding the inner working has been taken a step further, as the building block do not have any

connectors where generic modules can be connected and the standard game participant will not view the simulation model any more than a couple of minutes to see a simulation model run. The game participants cannot make adjustments to the simulation model in the simulation environment and they cannot change the functionality of a building block.

Simulation Building Block Guideline 11: advanced model developers have to be able to unhide the inner logic and see how the processes and attributes are implemented.

This option has not been provided to the model developers. See Simulation Building Block Guideline 10 for further explanation.

Guidelines related to providing useful services or functionality of building blocks

Simulation Building Block Guideline 12: system elements should be represented by building block elements that can be extended with custom instantiations of model constructs of a generic simulation environment.

This option has not been provided to the model developers. See Simulation Building Block Guideline 10 for further explanation.

Simulation Building Block Guideline 13: a building block can connect to model constructs of a generic simulation environment.

This option has not been provided to the model developers. See Simulation Building Block Guideline 10 for further explanation.

Guidelines related to precisely defined interfaces for building blocks

Simulation Building Block Guideline 14: the model developer has to adjust the parameters of a simulation building block via a user interface.

Each simulation building block has a user interface for parameter settings, but the game participants will not use the interfaces. They will use the drawing capability in VISIO or the database settings in the MS Access database (Figure 7.3). This can be seen as a dedicated user interface, though.

Simulation Building Block Guideline 15: use of domain terminology in the user interface provides insight in the suitability of a building block for a certain purpose and the meaning of its parameters.

The automatic model generation that is used in the visualization-simulation tool works the other way around than the Simulation Building Block Guideline suggests. If the VISIO drawing does not contain any of the following building blocks (road, storage, parking spot, crane or quay) the VB-program gives an error claiming that the drawing is incomplete. If all five types of building blocks are instantiated at least once in the drawing, the VB-program will make sure

that all information is set into the simulation model to result in a verified simulation model (Valentin et al, 2002).

The same applies for data entry in the MS Access database, if no trucks are available or no companies are selected, the simulation model will not be populated with data, but the game participant will be informed of his/her omission.

Simulation Building Block Guideline 16: parameters in a user interface of a simulation building block have to be checked for validity of the values.

The MS Access database contains field controls to check that valid data is entered. The VISIO drawing is limited to the building blocks provided in the stencil in VISIO.

Simulation Building Block Guideline 17: parameters in a user interface of a simulation building block should have default values whenever possible.

Again, because we are working with data from the MS Access database that is checked for logical values, absence of this Simulation Building Block Guideline does not result in any trouble for the game participants.

Simulation Building Block Guideline 18: The user interface of a simulation building block should provide support for the model developer.

The game participants do not need support in developing the simulation model or setting the parameters of the building block, they need support in making a drawing in VISIO. With the example at the beginning of the game and the familiarity of most game participants with VISIO, this Simulation Building Block Guideline is matched.

Simulation Building Block Guideline 19: The user interface of a simulation building block can be used by model developers to select building block elements and set their parameters.

This Simulation Building Block Guideline is not applicable, because building blocks are completely fixed and do not contain alternative building block elements.

Simulation Building Block Guideline 20: a simulation building block has a defined interface that receives triggers, requests, entities, or events from other simulation building blocks in the simulation model and redistributes these internally.

This Simulation Building Block Guideline is handled by the coding for the status monitoring of the container entity.

Simulation Building Block Guideline 21: the interface of a simulation building block contains evaluations of the state of the trigger and the building block to determine whether the building block can handle the trigger.

The status monitoring is used mainly for debugging purposes during the development of the domain specific extension and its building blocks. With completely closed building blocks and the automatic model generation the Simulation Building Block Guideline is no longer is applicable for the case study.

Simulation Building Block Guideline 22: a simulation building block contains pictures, numbers and other elements to support visualization of the state and key performance indicators during simulation run.

The Arena building block screenshots in Table 7.2 show the visualization the game participant can see during the model run, but in the visualization-simulation tool the need for these interfaces is not as large as in other simulation studies. The reason is that the game participants mainly ran the simulation model without animation. They directly received the outcome of their design in the Excel sheet and thus usually skipped the phase of observing the model animation.

7.6 Simulation studies performed

In the designed game, 10 different actors have 8 hours to come up with a design of an inland container terminal. The challenge of the participants was twofold, 1) design a terminal that all participants agree on and 2) design a process for further negotiations and arrangements. The participants in the game had different aims for the container terminal regarding size, location and safety. On purpose some of the potential desires could not be satisfied with the visualization-simulation tool, for example, the visualization-simulation tool does not include the cost of moving the current garbage retrieval side of the municipality, which is at the location where the container terminal has t obe built, to another location. If the participants stumbled on this or similar issues during their design process, they had to address this in their future process of negotiation and arrangements (Bockstael et al, 2003).

The participants in the game had four workstations available for all participants, independent of the stakeholder they were representing. At the workstations the participants could work on the design of the inland container terminal. In each game the use of the tool was slightly different as the game participants were free to organize themselves as they thought to be best suited. In most games the participants decided to divide themselves in three groups with representatives of each of the available teams, one group discussing the future process, one group developing a terminal as large as possible and one group developing an environmentally friendly solution. The groups came together to discuss their progress and show their intermediate

results at different moments during the game. The participants often had their first terminal design that made some profit after one hour and from then onwards they worked on optimizing the design according to their desires, e.g. increase the size or eliminate some types of customers, aiming at profit optimization.

The game has been used in different settings. Initially, the game has been played in three gaming-simulation sessions with 77 participants, who were students who signed up for a voluntary part of a course in the TU Delft's Technology, Policy and Management third year curriculum. Secondly we played the gaming-simulation three times with 20 to 30 MBA-course managers and thirdly the game has been played with 65 students again from the Faculty of Technology, Policy and Management at TU Delft. These sessions have been evaluated using questionnaires and external observers (Bockstael et al, 2003). In total the management game has been played for more than 20 times by bachelor students, students in MBA courses and experts in process management.

The participants in the game needed only a little bit of support in explaining what performance indicators meant and what conclusions they could draw from the results in the Excel interface. They succeeded with very limited training in building their own terminal, export the configuration to the simulation environment Arena and perform simulation experiments. They worked their way through the results in the Excel interface and from there they evaluated their design and identified possible improvements. Every experiment was performed after some discussion between the different stakeholders about the suitability of an experiment. For example, if the participants analyzed that the storage space should be enlarged, then they first had a discussion where the extension should be located, including the height and the shape of the storage. Once the discussion was finalized, the design was adjusted in their VISIO drawing and a new simulation run was performed.

The types of experiments that the participants in the game performed were all along the lines of the capabilities of the visualization-simulation tool. Experiments that were not supported, for example evaluating an alternative lock at the entrance of the port, were scheduled for further analysis in the next phase of the decision making process. As a result the experiments performed with the visualization-simulation tool aligned perfectly with the interface capabilities in the VISIO drawing or the database.

In all the performed games the participants, none of them simulation experts, succeeded in developing two or three different terminal designs that they would like to enhance in further evaluations. In most of the gaming sessions the participants had time to improve their design and increase the expected profit. In games where they did not succeed in optimizing, this was mainly caused by extensive discussion before they started to use the visualization-simulation tool.

7.7 Observations during simulation studies

The initial game session have been evaluated extensively (Bockstael et al, 2003). Empirical data were gathered through: (1) observation of the group work during the game; (2) plenary debriefing after each game; (3) a written questionnaire filled out by the participants shortly after the game; (4) a written response by the participants to a few open questions. The evaluation of the visualization-simulation tool and the game sessions focused on three main aspects: (1) the role of the visualization-simulation tool in an inter-organizational decision-making process; (2) the generation of some specific requirements for using such a tool; (3) some suggestions for refining the game and the visualization-simulation tool for real life systems. The evaluation criteria regarding the visualization tool and the gaming-simulation were generally derived from notions on infrastructure design in inter-organizational settings.

Bockstael et al (2003) and Mayer et al (2004) describe the results of the questionnaires and observations mainly from the process management viewpoint. They focus on the function of the visualization-simulation tool as part of the game and the design of the game to teach students the interaction and use of quantified data in a complex multi-actor design process. They show that the students in majority found the visualization-simulation tool helpful and that they could add quantified performance indicators to discussions in an early stage of the design process.

Valentin et al (2002) conclude that the use of the visualization-simulation tool showed that the use of simulation does not need to be something to be postponed until the final design phase. The participants in the game could easily and quickly make a representation of their idea and evaluate the effects, due to the support of the complete model development cycle (Figure 7.1). Without the automatic model development and collection of performance indicators the simulation model would not have been of any use in the design phase, and without the building blocks of the domain specific extension for container inland terminals these tools could not have been developed.

7.7.1 Observations regarding design approach and implementation

Every decision made during the design of the domain specific extension and its implementation in the simulation environment Arena was aimed to enable fast and easy model development. For example, all companies were located 20 kilometers from the container terminal resulting in fixed cost for transport of containers and no distinction between companies. This was just one of at least a dozen assumptions that were made in the design of the domain specific extension, as it was designed for this specific use only and not for evaluation of real inland container terminals.

7.7.2 Observations regarding additional tools

The additional tools have had their issues during the development process, mainly the automatic connectivity between VISIO, MS Access, Arena and Excel, but the second release was a good improvement. Many of the connectivity problems were resolved. The availability of the additional tools has been a key success of the domain specific extension and the success of the game. The additional tools made the simulation model into a magic black box that provided answers to the questions of the game participants regarding a design, exactly what the game needed.

7.7.3 Observations regarding provided support

The game participants were not hindered by the lack of knowledge about the building blocks or the use of the domain specific extension. The game participants studied the support material, but mainly to use it as a judgment of the quality and the validity of the visualization-simulation tool. Given the scope of the game, the provided support was a good match.

7.7.4 Observations regarding applying building block concept and guidelines

Even though the need to update and extend the set of building blocks in this case study did not occur, we still made observations regarding the Simulation Building Block Guidelines. The observations for the capabilities of the domain specific extension are structured using the characteristics of a building block as defined by Verbraeck et al (2002).

Table 7.3: Characteristics of building blocks in Container Terminal game

Self-Contained	Positive	the container stored the information and the outcome of decisions in the other blocks.
Interoperable	Positive	centralized control versus individual physical elements and container kept information and pointers to actors.
	To be improved	no family structure and no process description of the control elements, as both were seen as not necessary for the type of experiments.
Reusable	Positive	the domain specific extension has been reused many times, although it always was to model the same system, but every time in a slightly different session with different users.

(continued at next page)

Replaceable	Not applicable	the building blocks were not replaceable and no use of building block elements other than structuring. Furthermore there was no need to extend the set of building blocks or building block elements.
Encapsulating its internal structure	Positive	the additional tools completely hide the simulation model, including the inner logic of the simulation building blocks.
Providing useful services or functionality	Positive	the system of a container terminal was completely composed out of simulation building blocks.
	To be improved	the simulation building blocks could not be extended or adjusted to provide additional services or functionalities.
Precisely defined interfaces	Positive	parameters of the user interface were automatically defined and technical interfaces between the building blocks were correctly aligned.

7.8 Overview observations

7.8.1 Observed benefits in management game for container terminal

Table 7.4 provides a similar summarizing overview as presented in chapter 3 tables 3.6 and 3.7, regarding only the benefits. Similar to in the tables in chapter 3: If “No” is filled in for a potential benefit it means that in the case studies for this domain we did not observe effects of the expected benefit. This is not negative, but points out that the case study has the potential of being even more effective.

The benefits for the domain specific extension ‘Container Adrift’ game range from a very explicit yes (‘Yes !’) to a ‘Partly’. The explicit ‘Yes !’ is a result of the ability to perform a complete model cycle in 15 minutes. The ‘Partly’ is a result of the way the game participants used the visualization-simulation game. The game participants could have a look at the visualization of the simulation model, but they hardly used the opportunity, as they preferred to directly use the overview of performance indicators provided via the additional tool in Excel (Valentin et al, 2002; Bockstael et al, 2003; Mayer et al, 2004).

Table 7.4: Summary of benefits observed in case study Container Terminal

Process step **Expected benefits as mentioned in chapter 2 and 3**	**Observation container**
Activity 1: Problem description & define conceptual model	
Benefit 1.1: conceptualize system elements with model constructs in mind	Yes
Activity 2: Select model constructs	
Benefit 2.1: no translation between system elements and model constructs	Yes
Benefit 2.2: compose model constructs from developed domain specific model constructs to represent system elements	Yes
Benefit 2.3: easy selection of model construct thanks to structure of domain specific extension	Yes
Activity 3: Data collection	
Benefit 3.1: collection of predefined input data	Yes
Activity 4: Instantiate simulation model for original system	
Benefit 4.1: less model constructs used	Yes
Benefit 4.2: model development faster and easier	Yes !
Benefit 4.3: model development by simulation novices	Yes !
Activity 5: Verify and validate simulation model for original system	
Benefit 5.1: no more detailed testing	Yes
Benefit 5.2: easily gathering validation data	Partly
Benefit 5.3: structured and standardized performance indicators	Yes
Benefit 5.4: semi-automatic reporting of performance indicators	Yes !
Benefit 5.5: observe animation at different levels of the composition: high level and at individual model construct	Partly
Activity 6: Analyze output of simulation model	
Benefit 6.1: structured and standardized performance indicators	Yes
Benefit 6.2: semi-automatic reporting of performance indicators	Yes !
Activity 7: Define solution for analyzed outcome	
Benefit 7.1: model developers are triggered to find new solutions by parameters	Partly
Activity 8: Instantiate simulation model for identified solution	
Benefit 8.1: easy adjustment of model thanks to user interfaces of model constructs	Yes
Benefit 8.2: easy adjustment of model thanks to replacement of model constructs	Yes
Benefit 8.3: easy visualization thanks to incorporation of visualization in model constructs	Partly
Benefit 8.4: composition of new model constructs enabled new solutions to be evaluated	No

7.8.2 Observed risks in domain Container Terminals

The execution of the management game also provided some observations how the risks introduced in chapter 2 and 3 have been mitigated in the use and further development of the domain specific extension for container

terminals. The Simulation Building Block Guidelines and design approach of chapter 5 enabled to avoid most of the risks described, especially the ones regarding time consuming execution, thanks to the many automated processes applied in this case study.

Table 7.5: Summary of risks observed in case study Container Terminal

Process step **Potential risks as mentioned in chapter 2 and 3**	**Observation container**
Activity 1: Problem description & define conceptual model	
Risk 1.1: scope of model developer is limited by model constructs	Partly
Activity 2: Select model constructs	
Risk 2.1: lack of trust results in no motivation to use domain specific extension	No
Risk 2.2: lack of insight in model constructs results in ignore domain specific extension	No
Risk 2.3: use of model constructs that are not suited for representation of system elements	No
Risk 2.4: system elements can not be represented by model constructs	No
Risk 2.5: compose model constructs from developed domain specific model constructs only applied for infrastructure system elements	No
Risk 2.6: model developers can adjust internal logic of model constructs	No
Activity 3: Data collection	
No risks defined in chapter 2 or 3	
Activity 4: Instantiate simulation model for original system	
Risk 4.1: model developers do not understand model construct	No
Risk 4.2: model developers do not know how to parameterize model construct	No
Risk 4.3: difficult to compose simulation model, because model constructs are not available	Partly
Risk 4.4: difficult to compose simulation model by person other than developer(s) domain specific extension	No !
Activity 5: Verify and validate simulation model for original system	
Risk 5.1: mistakes of model developer are hard to overcome	No
Risk 5.2: model developers know something is wrong, but cannot identify what to do about it	No
Activity 6: Analyze outcome of simulation model	
Risk 6.1: model constructs do not provide performance indicators problem owner desired	No
Activity 7: Define solution for analyzed outcome	
Risk 7.1: model developers are triggered to find new solutions by parameters	No
Risk 7.2: model developers are limited by parameters and model constructs	Partly

Activity 8: Instantiate simulation model for identified solution	
Risk 8.1: solution is identified that cannot be represented by model constructs	Partly
Risk 8.2: adjustments of model constructs required to represent solution are time consuming	No
Risk 8.3: replacement of model constructs causes errors in model constructs that were linked or connected.	No

The game participants in the game 'Containers Adrift' observed mainly risks that had to do with the scope of the visualization-simulation tool. The game had a broader scope than the capability of the visualization-simulation tool, therefore it was logical that game participants would have further questions. This game aspect was added on purpose, to teach the game participants that not everything can be known and decided in the first initial design, but that decisions on the container terminal might be postponed to later phases of the negotiations. These risks thus have been encountered, but on purpose as part of the game design; therefore the observation is marked with 'Partly'.

8 Application to Nestlé production facilities

8.1 Why develop a domain specific extension?

Nestlé produces food products worldwide. Their product range consists of many different types of products and includes milk powders, ice cream, yoghurt, chocolate, water, soup and pet food. The products are aimed at the top segment of the market, and are distributed in the wide region around the factory. Distribution is not limited to the country where the factory is located. The manufacturing processes used at Nestlé are aimed at producing a wide range of high quality products for an extensive range of customers.

Each Nestlé factory deals with different challenges, but in general the factories strive for high efficiency on the production and the packing lines, taking into account the need for effective and efficient changes when changing from one product to another within the production system. Improving the efficiency of a production cycle, for example by producing larger batch sizes, will have negative effects with respect to required storage space and product longevity. Further, Nestlé's R&D departments are constantly inventing new products. These new products require alternative production process and different (new) packaging materials. More types of products, produced within one factory, with different package sizes results in factories where there is a high level of changeovers for products and packing lines.

The design of any new factory is based on the product mix expected to be produced once the factory is completed. However, new products or expansion plans lead to an adjusted product mix. Consequently, the factory management needs to find new and alternative means to meet the new production demands and market requirements. The factory management will stretch the production capabilities of their factory as much as they can, but eventually, investment in a factory might be required to meet the requirements of effective production of the increased product mix. Some of Nestlé's factories are more than twenty years old and over time Nestlé has increased product throughput by 50 to 200% when compared to their original expected throughput. However, not all Nestlé's investments in their factories have resulted in the efficiency and throughput improvements that management expected.

Nestlé process engineers have few tools they can use to evaluate whether their suggested improvements will pay-off before implementation, and result in the desired production capacity. To deal with this problem the process engineers were considering using simulation technology as a means to achieve three goals:

1) Improving the design of factory
2) Supporting operational planning
3) Improving and standardizing processes, and planning and testing product automation ideas before implementing changes on the factory floor

Nestlé's different departments do not have sufficient workload to make hiring and educating full time simulation model developers economical. They have a quickly rotating team of process engineers who will use simulation tools for 1 month in a project at a factory and then for 7 or 8 months do something completely different. Educating these process engineers so that they can develop valid simulation models in such an environment would be a waste of their other talents and time. A domain specific extension specifically designed for Nestlé and its batch production technology that can be used by the process engineers would resolve this issue.

Nestlé started a couple of simulation projects in 2004 to evaluate the use of simulation tools and different generic simulation environments in the design of a factory. The ultimate goal was to enable the process engineers to construct a simulation model of a factory, to perform an analysis and provide suggestions for improvement. The simulation models should be able to be used by the operational planning departments and to interact with the SAP Globe ERP system and the PLCs of the physical equipment used in the factory. In 2005 Nestlé Nutrition selected Systems Navigator and Rockwell Software to use the Arena simulation environment to develop a domain specific extension.

8.2 Initial team to develop the domain specific extension

The domain expertise for the initial domain specific extension was provided by a team of product engineers from the R&D center of Nestlé Nutrition. Together with simulation experts from Systems Navigator and automation engineers from Rockwell Software they defined the scope and problem domain for the domain specific extension.

The simulation experts of Systems Navigator completed the development of the set of simulation building blocks, including example simulation models and teaching material. The initial simulation studies have also been performed by simulation experts of Systems Navigator, supported by subject matter experts from Nestlé trained with the use of the domain specific extension.

8.3 Specification of the domain specific extension

8.3.1 Scope

The scoping of the domain specific extension was mainly based on the experience gained during two simulation studies, one for a fresh milk factory in Asia and another for a milk powder factory in Europe. These initial

simulation studies were performed to verify assumptions regarding the level of detail of the simulation models, the performance indicators, and the technical implementation. The conclusion of the simulation studies was that the following assumptions were valid to apply in the design of the domain specific extension:

- Pipes are not a constraint for the production process, neither is the cleaning of these pieces of equipment. Given the complexity that of adding them to the simulation model and the lack of availability of the design of the piping network at the stage of the design of the production facility, it is better to keep them out of the simulation model.
- The real factory has production and flow rates with a variance. The operators have the task to adjust the valves and monitor the equipment to keep the production rate as close as possible to an optimized rate. Optimized rate in this case means 'optimized for the quality of the product'. The simulation models applied a constant flow rate per recipe and this provided sufficient insight into the overall production process.
- The production process has a slowly increasing rate during the startup of the process for a certain recipe. The production of goods during this period of time is kept out of scope, only the quantity produced during stable running of the production is considered. The startup and the cleaning activities are represented by a fixed period of time that the equipment is not capable of producing.
- Human resources are available and are not considered to be a bottleneck for the production process.
- The performance indicators used in the simulation studies are sufficient for the scoping of the domain specific extension.
- The generic concept of Arena tanks and flows is suited for the modeling of Nestlé production facilities.

The learning points of the simulation studies and the approved assumptions have been used by the simulation experts to further describe the scope and problem domain of the domain specific extension. Before the document that described these elements was presented to Nestlé, a workshop was organized to achieve further awareness and buy-in from Nestlé's process engineers. In a workshop with 20 experts from the Nestlé Nutrition R&D center, Figure 8.1 was discussed. Some of these 20 experts had been involved in the initial simulation studies in 2004. In pairs of two the experts made adjustments to the drawings to point out experiments they would typically like to perform and what kind of feedback they expect from the simulation model to judge the suitability of the adjustments.

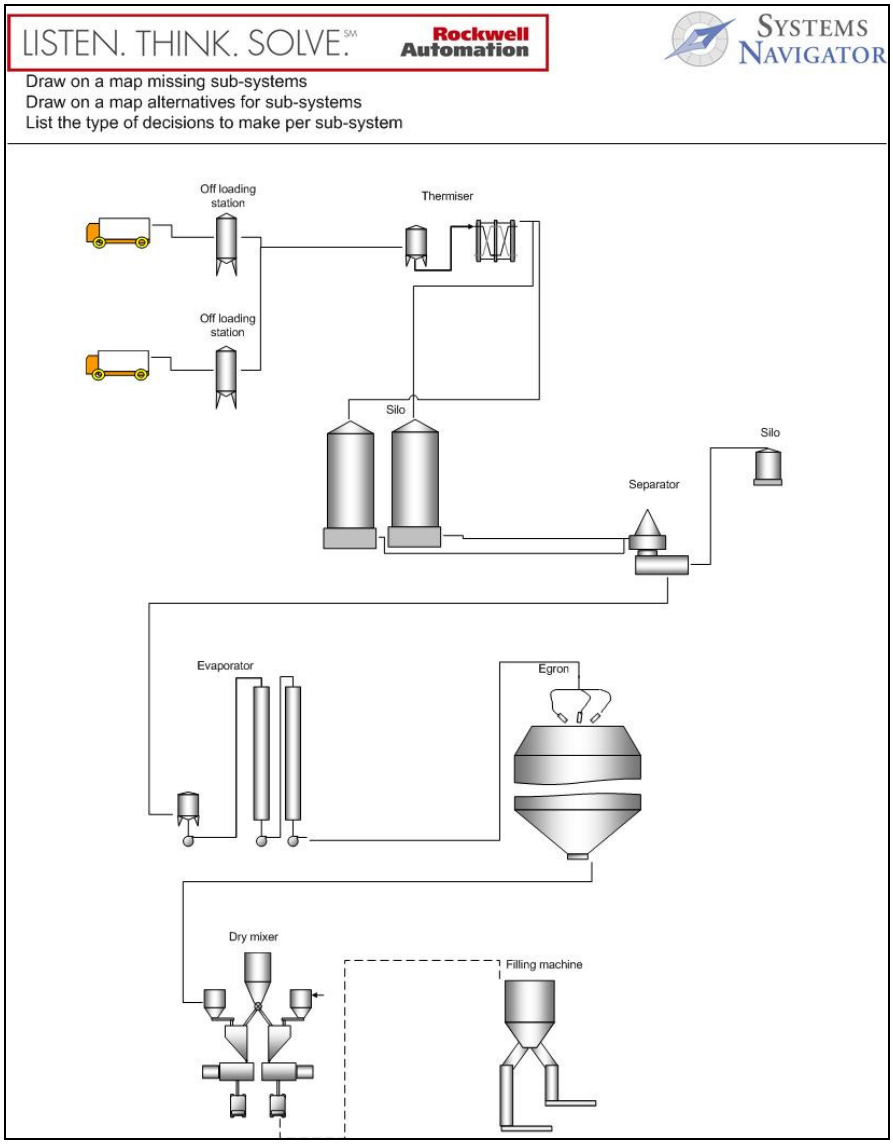


Figure 8.1: Example factory to trigger discussion on experiments and scope (Spruengli et al, 2005)

The results of the drawings were discussed with the full group. Spruengli et al (2005) describe the full listing of the identified experiments. Some example factors or variables of the experiments identified by the process engineers are:

- Different quality levels of the fresh milk accepted by the trucks
- Change the number of tanks for storage
- Change the size of tanks
- Change the rotation rate of the centrifuge
- Different cleaning schemes of the centrifuge
- Use of the skimmed milk storage, or direct flow towards evaporators
- Change the maximum storage time of product in tanks
- Recipes depending on evaporator production rates

- Using inline mixing before evaporator or even after storage in powder bins
- Alignment of production scheme (and cleaning intervals) of evaporator with spray dryer
- Dry powder storage in bins, totes or bags
- Adapting the pack type settings of the filling machine
- Varying the number of filling machines
- Changing the production rate of filling machine

The process engineers typically followed the process of the production and identified the issues they are struggling with in each of the particular production departments. Spruengli et al (2005) have structured the types of experiments accordingly:

- Experiments regarding number of equipment
- Experiments regarding parameter settings of equipment
- Experiments with process sequence and dependencies
- Experiments with recipe and product scope
- Experiments with planning and scheduling

The reporting that the process engineers expected could be structured according to the same lines, but it was important to notice that the priority of the performance indicators was in reverse order:

- Quantity of product produced by the production facilities
- Lead time of product in storage from first drop to empty storage
- Production stoppages due to lack of product
- Energy consumption for operating the factory
- Fill rate of storage tanks
- Utilization of equipment

Figure 8.1, which was used in the workshop to identify the experiments, resembles a small factory. Nevertheless, the experts in the workshop concluded that the experiments that they identified and the performance indicators that they claimed to be interested in were also applicable for larger factories. Whether a factory was producing 5 or 25 different recipes and 10 or 150 different finished products results in differences for the complexity of the planning, but the questions that the factory management of large sites have are the same.

The expertise of the process engineers resembled a broad overlap of areas within Nestlé production facilities. Some of the process engineers were mainly involved in the Nestlé Nutrition factories of the future, a greenfield concept with minimum intermediate storage and less dependency on fresh milk. Other process engineers were specialized in improving the production facilities of existing factories where they had to deal with a lot of existing equipment and processes. The combined knowledge provided a large list of experiments, with many overlaps (Spruengli et al, 2005).

8.3.2 Problem domain

The physical objects in the Nestlé production environments were mainly defined in the workshop with the process engineers to identify the scope in combination with the equipment for the initial simulation studies. The list of objects include milk intake station; fresh milk storage tank; centrifuging unit; inline mixing equipment; batch mixer; dissolver; evaporator; spray dryer; powder storage; powder filling machine; liquid filling machine; cleaning equipment. The objects were put in a hierarchy and the top of the hierarchy for all equipment, except for the cleaning equipment, is a piece of equipment that contain product (liquid or powder) that flows in and/or out. The most generic equipment in the top of the hierarchy are the storage silo or the powder bin. The full hierarchy of physical elements in Nestlé production facilities is described in Valentin et al (2005a).

The hierarchy of equipment could be extended much further than only the mentioned objects, but together with the process engineers of Nestlé it was decided to keep the following pieces of equipment out of scope of the object orientation: pipes, circuit stations, individual product addition mixers (e.g. inline sprayer for oil, honey or chocolate); storage tanks with specific functionalities, e.g. fermenters. The main argument was that if a system required this equipment to be included in the simulation model, the generic piece of equipment, i.e. a storage tank or a powder bin, can be used to represent these system elements with the addition of extra processes to represent the additional functionality to handle the product or recipe.

Nestlé organizes its factory according to a quantity of tonnage to be produced per recipe or per pack type. The equipment itself is not producing anything, unless an operator has an order to use the equipment to produce a certain quantity of a certain product according to a recipe configuration. The complexity of the system is therefore modeled by the processes of executing the order and not by the functionality of the equipment.

The processes consist of all the steps that need to be done to produce an order. The trigger for the process is an order coming from a production schedule or triggered by incoming trucks to deliver raw material that needs to be handled. The steps for an order to be performed are to select equipment to be used, claim equipment to avoid that several orders are mixed together, clean any remaining elements from the equipment from previous orders, fill the equipment with the product, stay in equipment as long as necessary to reach the desired quality, remove the product from equipment and possibly simultaneously evaporate some of the product, then release equipment for next order. The order might include several pieces of equipment in sequence to be executed, e.g. the example factory in Figure 8.1 had a sequence of centrifuging and evaporation, and an order might include parallel activities, e.g. the example factory in Figure 8.1 was active in parallel using the evaporator, inline mixing and spray dryer.

Figure 8.2, Figure 8.3 and Figure 8.4 show some alternative process flows used by Valentin et al (2005) to communicate with the process engineers of

Nestlé. Using these and other process flows the complexity and the individual sub-steps in the process flows have been identified.

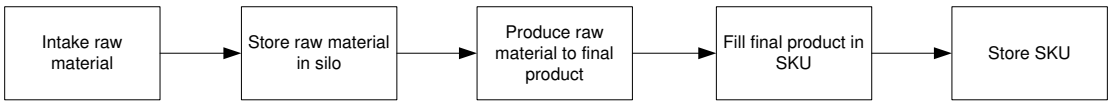


Figure 8.2: Process decomposition of complete process in milk factory

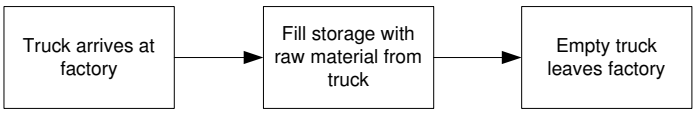


Figure 8.3: Process decomposition of milk intake process in Figure 8.2

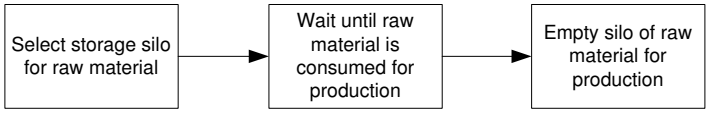


Figure 8.4: Process decomposition of store raw milk process in Figure 8.2

8.3.3 Building blocks

The need for individual equipment and flexible process descriptions was realized by defining three sets of building blocks that together form the domain specific extension for Nestlé Nutrition production facilities. The three sets were: building blocks for equipment, building blocks for process descriptions and control logic, and building blocks with definitions of recipes and production plans.

The separation in three sets of simulation building blocks was a preparation for the experiments that based on Spruengli et al (2005) can be summarized:

- Experiments regarding number of equipment and experiments regarding parameter settings of equipment are supported by the set of equipment building blocks.
- Experiments with process sequence and dependencies are supported by the set of process building blocks and the experiments with recipe and product scope.
- Experiments with planning and scheduling are supported by the definition building blocks.

The equipment building blocks were the elements identified in the workshop with the process engineers, but slightly more combined to reduce the set of building blocks. The key building block is the silo and in essence all equipment building blocks have been derived from this building block as already was defined by Valentin et al. (2005a) in the object hierarchy.

Figure 8.5 shows the physical interfaces that any type of equipment, whether a silo, an inline mixer or a spray dryer, encounters. These

interactions are handled by building block elements and the building block elements receive triggers from process building blocks which are shown in Figure 8.6. For example, the equipment ‘Silo’ has a building block element that generates triggers when the flow is stopped at the moment that nothing is entering or leaving the silo. The trigger is different if an incoming flow is stopped because the requested quantity is transferred or the maximum level of the silo is reached.

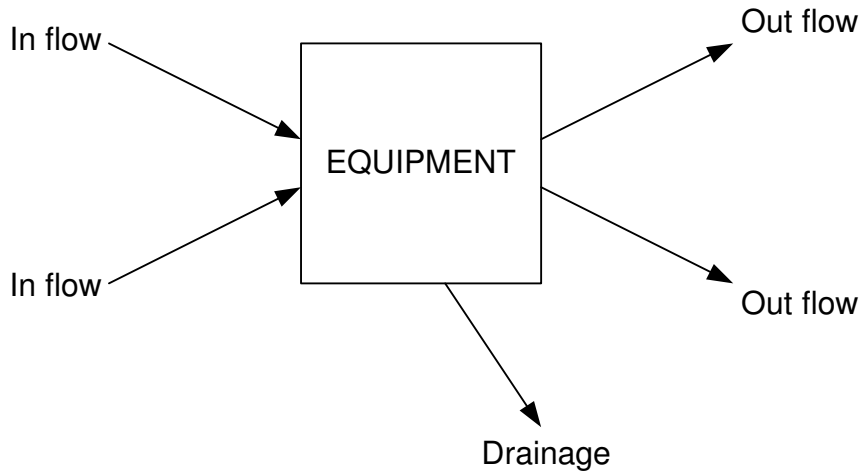


Figure 8.5: Example of physical interaction to a piece of equipment

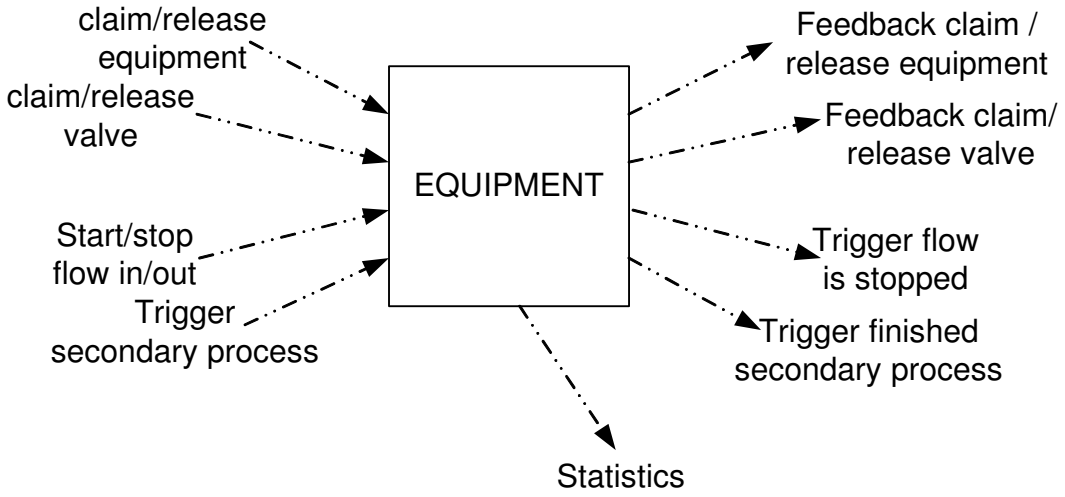


Figure 8.6: Logical interaction between piece of equipment and process description

The equipment will not perform any of the described activities of Figure 8.6 via a process building block. Each of the process building blocks provides a specific trigger for a function of equipment. As a result the set of process building blocks consists of ‘claim equipment’, ‘release equipment’, ‘transfer flow’ and ‘clean equipment’. In addition some process building blocks have been defined to overrule the state of a piece of equipment, for example the process ‘assign level’ overrides the current level in a piece of equipment

(mainly silo) to a new value. The complete list of process building blocks can be found in Valentin et al (2005a).

The process building blocks are related to one or two pieces of equipment and focus on one activity. In the process descriptions defined in the problem domain (Figure 8.2), several processes are combined that can be observed in several factories, and that could help in reducing the complexity of the simulation model. Examples of these advanced processes are batch mixing, evaporating, and the selection process. These more advanced process building blocks link different pieces of equipment and different process steps as described in Figure 8.6. These advanced process building blocks have been identified, but due to time and budget constraints these were not detailed at the initial phase of the project. In 2007 the domain specific extension was stabilized and the budget was made available to extend the set of process building blocks with these advanced building blocks. Valentin et al (2007) describe the advanced process building blocks that primarily were a composition of the existing process building blocks with limited parameters.

The set of building block with its definitions for recipe and production schedule is mainly used to provide a standardized way of documenting the recipes and production schedules in the simulation model. The advantage is the ability to use the structure in the parameters of the process building blocks and standardize a user interface in an additional tool. The content of these definition building blocks is provided in Valentin et al (2007).

8.3.4 Building block elements

The building blocks that contain product, such as silo, evaporator, spray dryer and centrifuge have a storage capacity and also contain physical building block elements to load or unload the product. These building block elements are called valves. The valves are necessary to enable processes to simultaneous handle in or out flows if it is permitted in the equipment.

The physical building block is composed out of physical and logical building block elements. The physical building block elements represent storage and incoming valves. The logical building block elements of the physical building block are the following:

- Claim equipment => allocate the equipment for a specific order or recipe.
- Release equipment => equipment is no longer in use.
- Claim valve => allocate a valve for a product flow in or out of the equipment.
- Release valve => earlier claimed valve is no longer in use.
- Clean process => reset the state of the equipment after a certain time period has passed.
- Statistics => collect data regarding quantity per recipe handled in equipment, time spent in a certain state (e.g. occupied or cleaning) and utilization of equipment, storage facility and available valves.

- Error generation => warning to model developer if equipment is asked to perform a process that cannot be performed due to its current state. For example an out flow while the storage does not have any content, a product flow in while the equipment is not claimed or a flow in for recipe X while the equipment is claimed to handle recipe Y. All these errors can be overridden with the parameters of the equipment to avoid that the simulation model stops for a permitted ‘error’. For example, recipe Y and X are different, but it is allowed to combine them together in the storage element.

The product containing building blocks also have building block elements that depend on the type of equipment. These building block elements have to do with the product handling of the equipment which is different for an evaporator or a centrifuge. The building block elements are:

- flow in for quantity X => Enable product transfer into the storage element of the building block. This building block element does not apply for a truck offloading station. The batch mixer building block has a variant that handles different recipes before a state is provided to perform a flow out. The inline mixer building block has a building block element variant that enforces that the flow in is for several recipe ingredients simultaneously.
- flow out for quantity Y => Enable product transfer out of the storage element of the building block. This building block element does not apply for a filling machine which produces SKUs at discrete time events and not a continuous flow. The batch mixer building block has a variant that checks whether the in-flow is finished. The evaporator, centrifuge and the spray dryer building blocks have a variant that also trigger the building block element ‘evaporize flow’.
- evaporize flow => Enable a secondary flow for building block equipment such as spray dryer, centrifuge and evaporator. The secondary flow is always transported to a storage unit (variant applied for centrifuge), can be transported to a storage unit (variant applied for spray dryer) or is lost (variant applied for evaporator).

The building block equipment is kept simple, therefore each piece of equipment has only one building block element per physical functionality that it represents. No alternative building block elements are defined. In fact, most of the building block elements that are standard in each building block (e.g. claim equipment, release equipment and clean process) are exactly the same for all equipment building blocks. In the design of the building block elements the assumption is made that if a system element requires a representation that is not provided by the building block that the model developer should handle this with the capabilities of the process building blocks.

The process building blocks also have building block elements that are standard for each building block and building block elements that vary depending on the process building block. Each process building block

consists of at least three building block elements, which are standard in all process building blocks:

- receive order => order enters the process building block and is registered to enable reporting and progress monitoring.
- depart order => order departs the process building block and monitoring of time and number orders handled is updated.
- statistics of order => report on the status of the process building block, like number orders still to be handled, number orders handled in total and processing time of orders.

The process building block has one or more building block elements that are activated between the building block elements ‘receive order’ and ‘depart order’. The building block elements between the ‘receive order’ and ‘depart order’ are comparable. The building block element of the process building block gives a ‘go’ to the selected equipment to execute the related function(s) of the equipment building block and its related building block element(s). After the process within the equipment building block element is finished the order returns to the process specific building block element for a wrap up process. Finally the order executes the ‘depart order’ building block elements and departs the process building block.

Figure 8.7 shows the generic sequence in a process building block. A typical example of this generic sequence is the process building block ‘Claim equipment’. The process specific building block element provides the order with the correct parameters to claim the equipment, i.e. equipment to be claimed, recipe to refer to and a code for routing (in this example code 1.01). The order will execute the logic of the building block element related to code 1.01, i.e. building block element ‘Claim equipment’. Once the code of the building block element within the equipment building block is fully performed the order will be returned to the process specific building block element of the process building block ‘Claim equipment’. Here some statistics of the claim process will be registered and then the order will depart the process building block via the ‘depart order’ building block element.

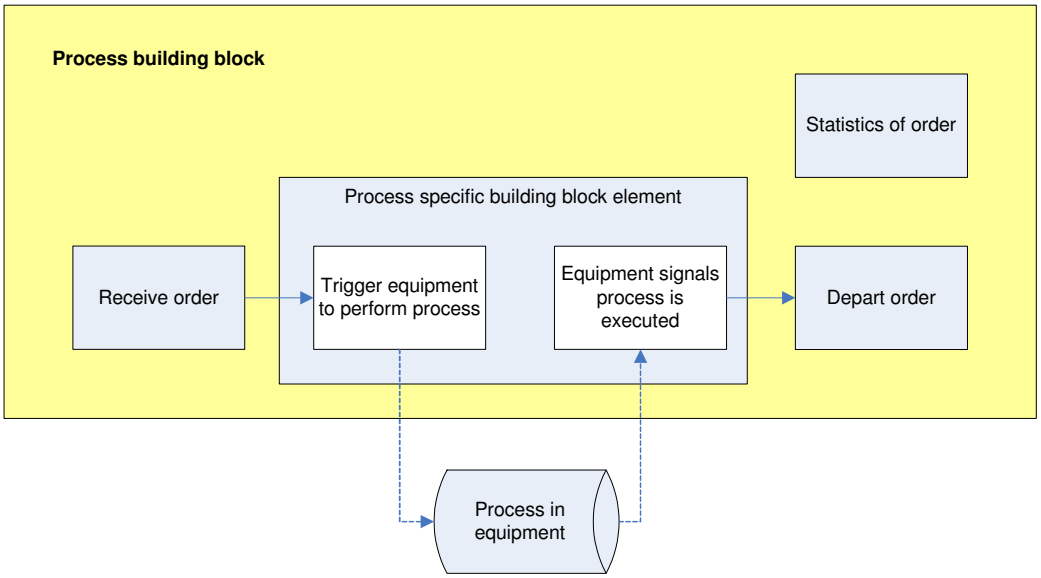


Figure 8.7: Sequence of processes in Process building block

Figure 8.7 and the example of the process building block 'Claim equipment' are simple process building blocks. More complex process building blocks contain several process specific building block elements that are handled sequential and/or parallel by the order. The process building block 'Transfer' is an example of a complex process building block that triggers processes to be performed sequential and parallel. Sequential the order prepares the physical equipment of the source of the product to be able to send the product (call the building block element 'claim valve' with code 1.10a) followed by preparing the physical equipment of the target of the product to be able to receive the product (call the building block element 'claim valve' with code 1.10b). Parallel activities are the triggers of the process building block for the physical flow between the source and the target via the 'flow out' and the 'flow in' building block elements of the equipment building blocks.

8.4 Implementation

8.4.1 Extension of simulation environment

The domain specific extension for Nestlé has been developed in the generic simulation environment Arena. The main reason is the fact that Arena is delivered by Rockwell Software, a company that is also responsible for a majority of the factory control software used by Nestlé. Nestlé thereby has the option to further integrate the simulation models with the factory control software by selecting the simulation environment that is provided by the same company.

The building blocks that are part of the Nestlé domain specific extension are divided into three sets. Figure 8.8 shows a part of the set of equipment building blocks and Figure 8.10 shows a part of the set of the process building blocks. The third set of building blocks relate to definitions and settings such as recipes and package types. Figure 8.8 and Figure 8.10 are not the initially

defined sets of building blocks, but they represent a part of the sets that were available in 2007. The set of equipment was initially developed for Nestlé Nutrition factories, and then extended with building blocks representing equipment used specifically in pet food factories (e.g. PetfoodMixer), coffee factories (e.g. Roaster) and ice cream and yoghurt factories (e.g. IceCreamFiller). These new building blocks follow the same principles described in Figure 8.5 and Figure 8.6. For example, the 'Roaster' building block contains all the same building block elements as the 'Silo' building block, except for determination of dedicated coffee process 'aging' in the roaster equipment.

Each of the equipment building blocks has a different visual representation that appears when the building block is instantiated in the simulation model. Figure 8.9 provides the example of the visual representation of the 'Silo' equipment building block. The bar inside the silo shows the relative filling grade of the storage building block element, the arrow shows the current flow direction (outgoing) and the numbers at the bottom give insight in the state of the building block regarding the current recipe, order and use.

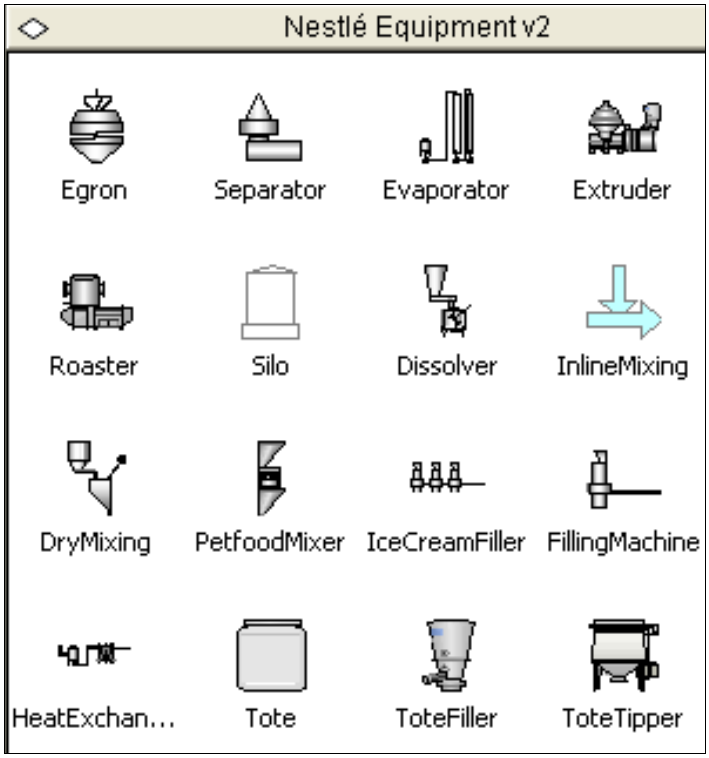


Figure 8.8: Part of set of equipment building blocks of Nestlé domain specific extension (Valentin, 2007)

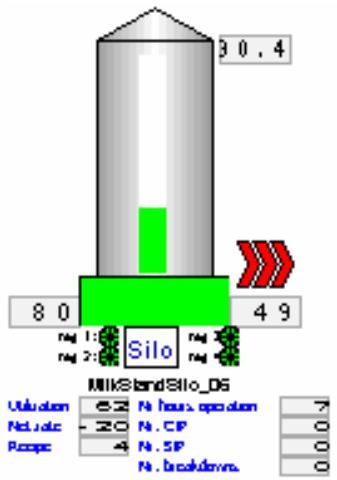


Figure 8.9: Visual representation of equipment building block 'Silo' during simulation run

The set of process building blocks was extended by performing projects in other environments than Nestlé's Nutrition sector. Process building blocks such as 'Claim equipment' and 'Filling process' are used in simulation models for all domains. In addition some processes were identified that are specific for domains at Nestlé. For example, the process building blocks 'PetfoodExtrusionProcess' and 'PetfoodMixProcess' are specific for controlling the equipment building blocks 'Extruder' and 'PetfoodMixer', both of which are only applicable to pet food factories.

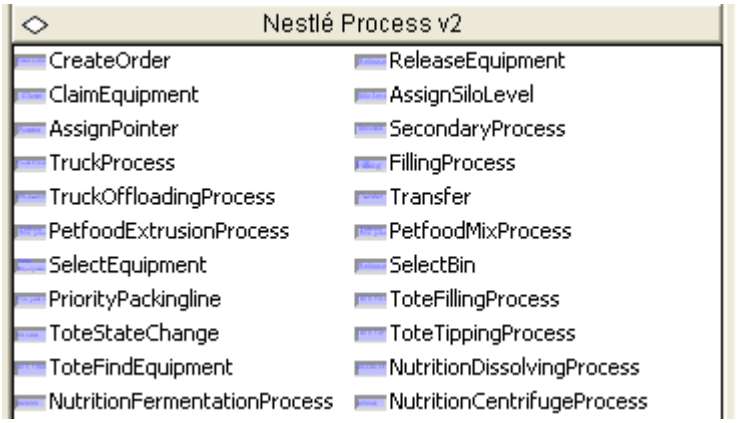


Figure 8.10: Part of set of process building blocks of Nestlé domain specific extension (Valentin, 2007)

The initial design of the equipment building blocks included a hierarchy and inheritance structure. Arena does not allow these kinds of structuring and Arena also does not allow instantiation of building block elements as separate modules within a building block. The building block elements are therefore completely implemented within the building block and the user interface for parameter settings and visualization of the state of the building block element is an integral part of the user interface and visualization of the building block. The development of the physical building blocks was performed by first

implementing the ‘Silo’ building block with all its building block elements and then duplicating this building block to represent the other building blocks of Nestlé Nutrition, e.g. ‘evaporator’, ‘extruder’, ‘centrifuge’ and ‘in-line mixer’. In the same way the set of equipment building blocks was extended for the representation of system elements of pet food, coffee and ice cream.

Figure 8.7 and the explanation around the process building blocks and their building block elements already described the use of coded triggers for building block elements of the equipment building blocks. Within Arena this is technically realized by defining unique stations for the equipment building blocks, followed by a decision tree that indicates which of the codes of a building block element has to be executed, similar to the routing mechanism used for Supply Chains (see chapter 6).

The applied design principle was to keep the equipment building blocks as simple as possible. This also included the availability of data of the equipment building block that enabled representation of its state. The applied rule of thumb was that all the information that an operator can see at his/her screens to make decisions regarding the process should also be available to the process building blocks in the simulation model. Therefore, most of the data regarding the state of an equipment building block were made public. Arena has a special tool for easily identifying the correct pointer to a data reference of a piece of equipment, called the ‘Expression builder’. Figure 8.11 shows an example of the expression builder used to identify the data reference of the current recipe handled in a silo building block.

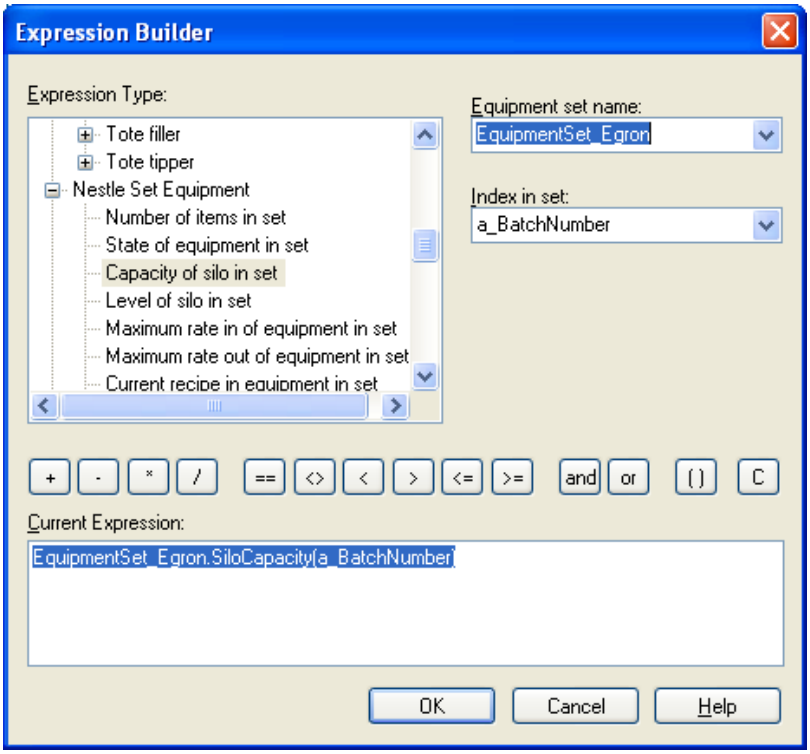


Figure 8.11: Expression builder for reference of building block

The Nestlé process building blocks interact with the equipment building blocks and evaluate their state to make decisions, while the Arena generic modules were used for further evaluation and handling of the entities. For example, in Figure 8.12 we can see the process building blocks for selecting equipment. These building blocks use the set of PetFoodMixer building blocks and search for a PetFoodMixer that is not in use. Additionally, an Assign and Delay module of the generic simulation environment Arena were used to keep track of information in the system and schedule an event to select another mixer or the same mixer again.

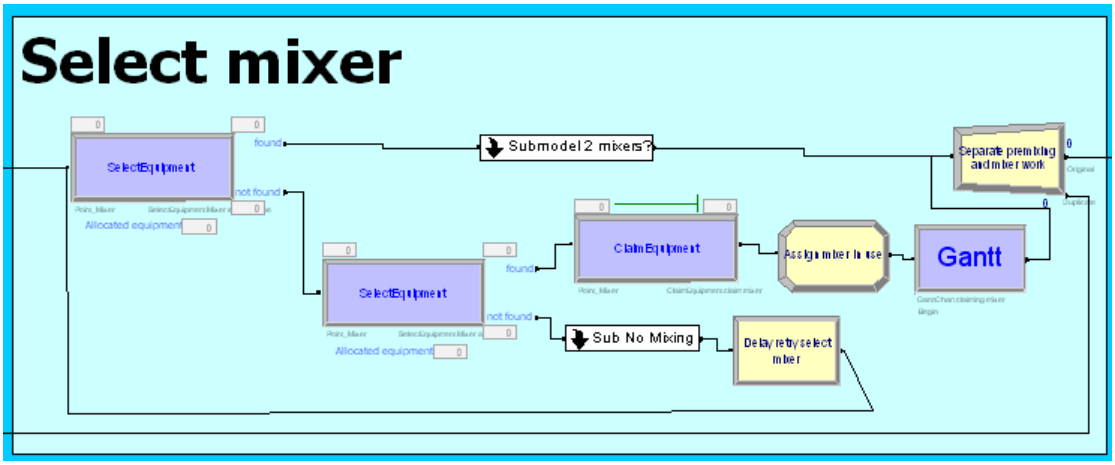


Figure 8.12: Mixture of building block processes and basic Arena modules in simulation model of pet food factory

The use of generic model constructs of the simulation environment Arena is also applied for exception handling when the simulation model should not give a user error, but handle the process accordingly. An example is a transfer from a source to a target that will trigger an error when the target is full and cannot handle any more quantity (see explanation underneath Figure 8.7). In the user interface of the process building block ‘Transfer’ the model developer has the option to stop the error from happening, and trigger a specific process in case the error occurs. The correct handling of the process of a full target destination is different in every factory, and even within factories the same trigger is handled in different ways depending on the state of the factory. For example, a trigger that a silo is full in a pet food factory in the UK can mean that the production is stopped, or if 95% of an order is completed it will indicate a need to start preparing the next order in a new silo. This variety of factory specific requirements cannot be implemented into building blocks of a domain specific extension, because the set of building blocks would grow exponentially with every new project. The concept we applied is shown in Figure 5.6 and as part of an actual simulation model of a Nestlé system in Figure 8.13.

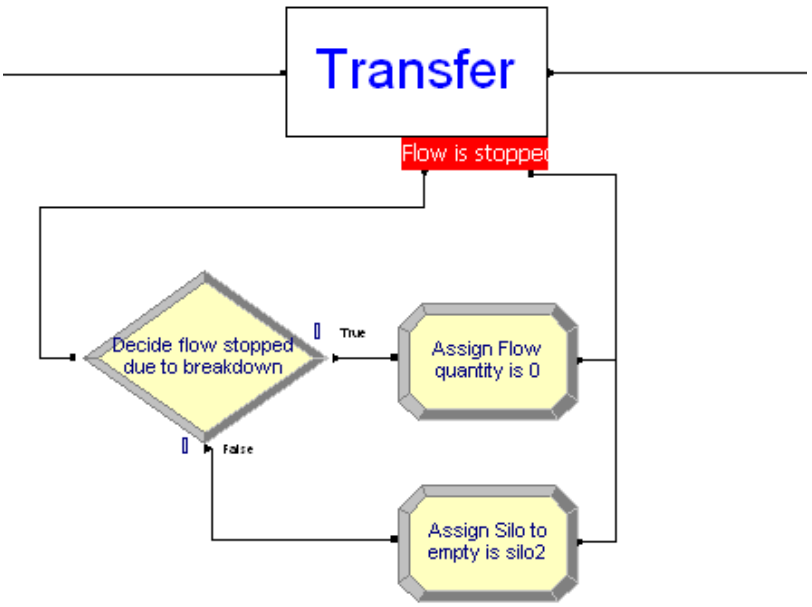


Figure 8.13: Example simulation code how to handle unexpected stop of flow

8.4.2 Additional tools

The domain specific extension for Nestlé's production systems is extended with an additional tool to support the generation, data entry and analysis of the outcome of the simulation models. The additional tool is an interface that easily can be configured to match a specific simulation model. The tool contains links to the user interfaces of the simulation building blocks that are instantiated in the simulation model for the equipment and the definitions of recipes and production plans. The tool further contains a mechanism to import the reports from the simulation building blocks and combine the different details into one overall sheet easily providing answers to the main question whether the production plan could be completely produced and if not, what the problem is. The tool was easily adjustable for a specific project. The tool was developed as an Excel interface that represents data for one scenario and as a Scenario Navigator (Gast et al, 2008) database that represents data from the scenarios analyzed during the simulation model run.

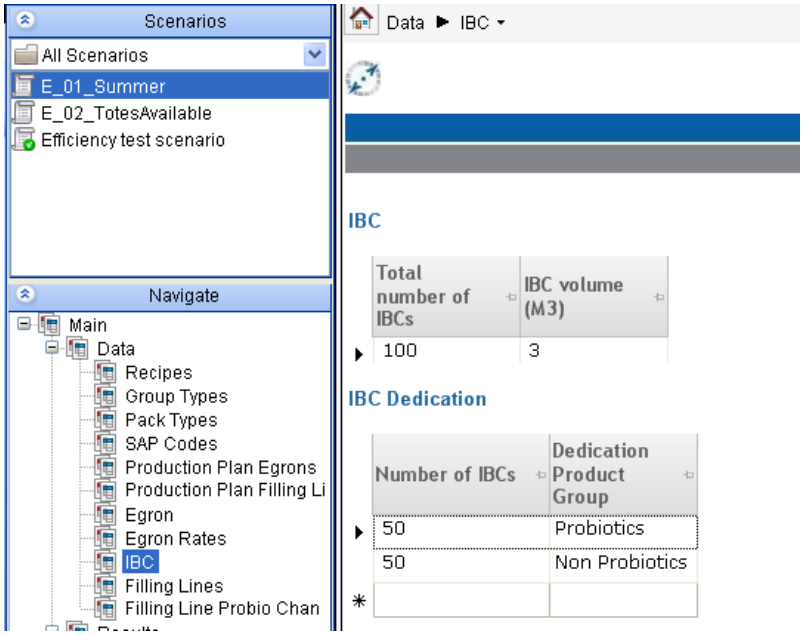


Figure 8.14: Input examples via Scenario Navigator for simulation model

Figure 8.15: Overview performance indicators via Scenario Navigator

The simulation model was interfaced via Scenario Navigator for input parameters, the collection of results and to trigger a planning algorithm. Input data for orders and availability of packing lines is shown in Figure 8.14. On the left hand side a list of scenarios is shown, each with its own data set of a performed or planned simulation. The key performance indicators of the selected scenario are shown in Figure 8.15. More detailed statistics can be viewed and evaluated by pressing buttons at the bottom of the screen. These

performance indicators are also available in reports and can be used to compare the results of different scenarios.

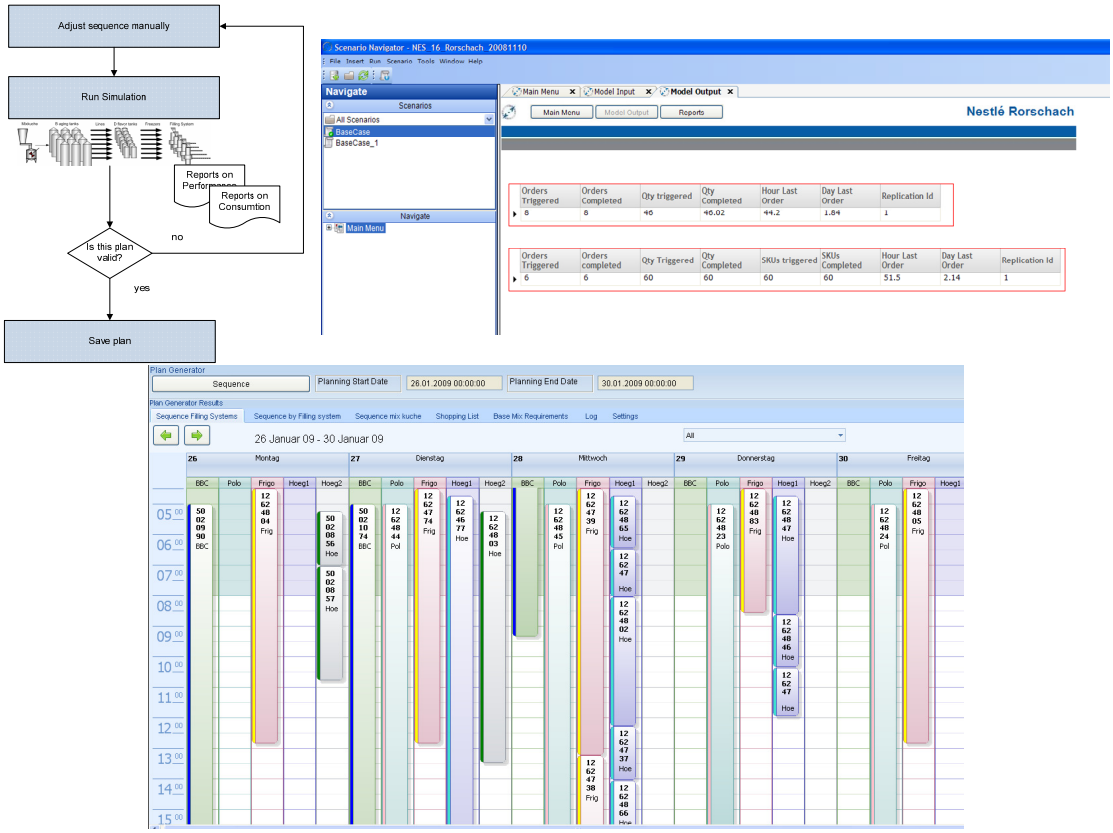


Figure 8.16: Extension to define production plan for simulation experiments

The processes in the simulation models are triggered by a production plan. In the simulation studies it turned out to be quite a challenge to define a valid production plan for a factory that is not yet in operation. In the simulation studies, every time the same evaluations were made to define the production planning, therefore an initial attempt has been made for the pet food factory in Hungary to define a production planning system based on the production requirements and the factory capability as defined in the simulation model of the factory. Based on the initial tool, called a ‘plan-generator’, a VisualBasic program has been made as an extension to Scenario Navigator that is capable of defining a production plan for a Nestlé factory based on data entry of the simulation experiment. The integration of the ‘plan-generator’ with Scenario Navigator enables the production plan to be used directly into the simulation experiment and also to enable easy experimentation with alternative production plans.

The custom made extension is shown in Figure 8.16 with the example of a pet food factory in the UK. This application evaluates the orders allocated to packing lines and defines a sub optimal plan for the extruders to feed the packing lines. A planner can manually improve the planning for the extruders

after a simulation run and, via several iterations, improve the extruder plan and reduce stock outs for the filling lines.

8.4.3 Support to users

The complete domain specific extension for Nestlé is much more than the three sets of building blocks discussed above. The domain specific extension was provided with extensive documentation for training and self-learning. A total of 75 small simulation models were developed to show the specific behavior of building blocks and combinations of building blocks. These range from a simulation model that shows how a quantity is transferred from one silo to another silo using the 'Transfer' process building block, to more complex situations like the selection of a silo to fill and empty the extruder process in a pet food factory. These small simulation models were used for several reasons: one, to test whether the building block is working correctly; two, to demonstrate that the building block is working; three, to demonstrate the possible use of building blocks in simulation models; four, as a teaching cases to introduce how to use the building blocks; and five, to test whether the building block still provides the agreed behavior after changes or extensions have been made to the building block or the set of building blocks.

The small simulation models have been described in the user manual of the Nestlé domain specific extension (Systems Navigator, 2007) together with the concept of building blocks and the user interface of all the building blocks for Nestlé. Version 2 (Systems Navigator, 2007) contains all the simulation building blocks and not only the initial sets for Nestlé Nutrition. The second version of the user manual has been part of a training package provided to 13 Nestlé process engineers from different domains including background information, example simulation models and assignments based on the small simulation models.

8.5 Use of building block guidelines

The previous case study described in chapter 7 aimed at quick model development with a very high level of abstraction. Realism and accuracy was not necessary in these simulation models and no adjustments were needed for the simulation building blocks of the domain specific extension to support different simulation studies. The requirements and the planned use by Nestlé is completely the opposite. The simulation models should contain a high level of accuracy and the simulation models should contain flexible logic, yet capable of handling complex decisions in the process of the system. Further the domain specific extension and its additional tools should be ready to be extended in any possible way:

- New infrastructure => new pieces of equipment within Nestlé Nutrition, but also support for equipment of other Nestlé domains, e.g. coffee, pet food and ice cream.

- New processes => alternative ways of operating a factory by reducing intermediate storage, producing new types of products, producing according to different rules or regulations and producing products completely different than milk powder, e.g. pet food, coffee capsules or yoghurt ice cream.
- Enhanced model development => Nestlé management assumed that every simulation model would be faster thanks to the gained experience, but more important due to the improvement of the set of building blocks and the additional tools.
- Enhanced model statistics => the performance indicator requirements of the domain specific extension were based on the initial simulation models for Nestlé Nutrition factories. Especially the simulation models concerning the pet food production process brought in new requirements to be able to analyze the reasons why the production process got stuck before the end of the complete production cycle.
- More model use => the simulation models were not only for greenfield systems, but also to improve existing systems and to provide suggestions for operational planning optimizations.
- New users => within the Nestlé Nutrition research and development center a process engineer stays only a couple of years and participates in several projects of which only one or two include simulation modeling. Therefore, new process engineers get involved in the model development. However, the analysts only look at one simulation model as they are only interested to analyze the results of their own factory. Therefore every new simulation model that is developed is analyzed together with a new group of experts.
- Enhance user friendliness => experiences gained in the model development of a simulation model and the analyses of the results via the additional tools had to lead to improvements of the user friendliness. For example, initially it was decided not to develop advanced process building blocks, in 2007 these advanced process building blocks have nevertheless been developed to replace the complete equipment selection by one simple process building block.

Guidelines related to self-contained building blocks

Simulation Building Block Guideline 1: data belonging to a building block should not be accessed by other building blocks directly, but only via defined interfaces.

The interface definition has been widened for the data availability of the equipment building blocks. Figure 8.11 shows how data could be retrieved via variables that are referred to anywhere in the simulation model logic, but the data is provided via a specific definition within the building block. Status data was thus available for the model developer outside the building block, because it was decided that the data exchange was open.

On the other hand, data that was not provided to the model developer via the Arena Expression Builder was not available outside the simulation building block. As a result, most of the statistics of process building blocks or statistics regarding production blockades were unknown for the inexperienced model developer until they were produced in the standardized reports of Scenario Navigator. On purpose we mention ‘inexperienced model developer’, because within the generic simulation environment Arena no data can be hidden, thus anything can be obtained by experienced users who are familiar with the inner working of the simulation building block.

Simulation Building Block Guideline 2: a simulation building block consists of a core and building block elements to represent functions and services.

The equipment building block and the process building block contain a lot of building block elements, which are structured and reused among the different building blocks. See the description in section 8.3.4.

Simulation Building Block Guideline 3: data belonging to a building block element can be accessed by other building blocks elements of that building block without using the interfaces of the simulation building block.

The implementation of the building block elements in Arena is included in the building blocks, whereby the boundaries of the building block element are set by the developer of the building block with some color boxes and texts. In the logic itself is not possible to distinguish building block elements. State parameters that are defined within one building block element, for example the state of a valve in the physical equipment building block element, can be evaluated by all logical elements in the building block without using further interfaces.

Guidelines related to interoperable of building blocks

Simulation Building Block Guideline 4: system elements that appear in different variants and processes in a system are represented by a family of building blocks and building block elements.

The clearest use of families in the domain specific extension for Nestlé production facilities is the use of three different sets: equipment, processes and definitions. The set of equipment building blocks can be separated into two families: the equipment that contain product (liquid or powder, or coffee, or pet food, or ice cream) and the equipment that is used to clean, for example the ‘cleaning station’, the ‘tote-cleaner’ or the ‘truck cleaning station’.

The process building blocks are all part of the same family that apply the same base structure as demonstrated at Figure 8.6. This family can be divided in sub-families:

- logic process building blocks => process building blocks applicable for logic process steps such as ‘Claim Equipment’, ‘Release Equipment’ or ‘Select Equipment’.
- execute process building blocks => process building blocks that execute a process with one or more pieces of equipment, for example ‘Transfer’, ‘Clean equipment’ or ‘Fill process’.
- specialized execute process building blocks => process building blocks developed to perform specialized processes with specific equipment of Nestlé Nutrition or Nestlé Pet food. For example, ‘Evaporate process’, ‘Pet food mix process’ and ‘Inline mix process’.

Simulation Building Block Guideline 5: building blocks are of different types, most common to have building blocks for infrastructure and for control.

The infrastructure and control guideline has been applied as way of working from the start of the project. The concept of simple equipment controlled by detailed and advanced processes has been one of the points that were verified with the initial simulation studies. This concept has been applied in the design of the building blocks since the initial simulation studies, resulting in the three sets of building blocks; equipment, processes and definition.

The control within the simulation models has been further improved by supporting the process building blocks with the full availability of the generic modules of the Arena simulation environment.

Simulation Building Block Guideline 6: complex control mechanisms should be represented using control building blocks linked together to represent a flow.

The need for this guideline already appeared in the discussion of the experiments with the process engineers (Spruengli et al, 2005). The process building blocks have one entry and two or more exits. The main aim of these connectors is to put the process building blocks in sequence and enable an order to be executed following the defined sequence. An example is shown in Figure 8.12 where several process building blocks are used in sequence to select which equipment to use regarding a mixing process.

Simulation Building Block Guideline 7: building blocks should be aware of each other’s existence within a range of applicability.

The references to equipment between process building blocks and equipment building blocks is achieved in two different ways, both to be defined in the parameters of the process building blocks as shown in Figure 8.17. The left user interface shown in Figure 8.17 represents a fixed link between the process building block and the equipment by referring to the name of the equipment selected from a pull down list with all equipment instantiated in the simulation model. Every entity that enters this process

building block will be triggered to perform a logic for a secondary process same equipment building block, in this example ‘Filling_Can_1’.

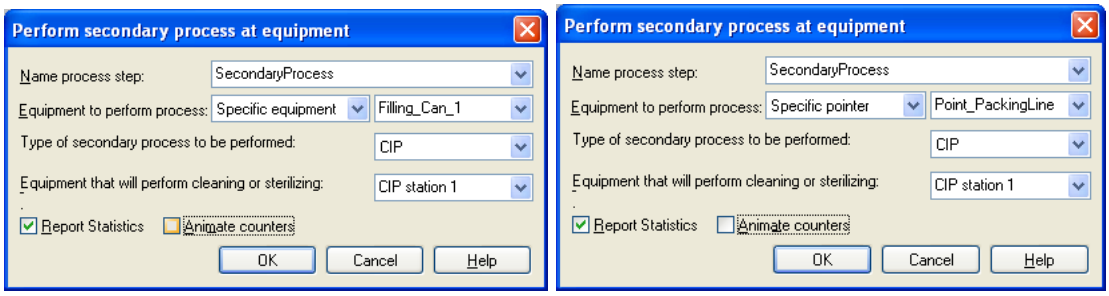


Figure 8.17: User interfaces of process building block 'Claim equipment' (left=reference via fix name; right=reference via pointer)

The right user interface shown in Figure 8.17 uses a dynamic link where an order previously has received an attribute that contains a pointer to an equipment building block. Within the process block the order will be sent to execute the process to a previously assigned equipment building block of which the reference is stored in the pointer ‘Point_PackingLine’. The pointer could have been set in several ways, most common is the use of the process building block ‘Assign pointer’ in which a specific equipment is allocated to the pointer, or the use of the process building block ‘Select Equipment’ in which a suited piece of equipment is allocated based on conditions and a set of equipment.

Guidelines related to replaceable unit of building blocks

Simulation Building Block Guideline 8: extension of a domain specific extension can be achieved by introducing new building block elements for existing simulation building blocks.

The initial set of simulation building blocks for equipment focused only on Nestlé Nutrition. New building blocks that have been developed are for example the ‘extruder’, ‘grinder’ and ‘pet food mixer’ for pet food factories, the ‘roaster’ for coffee factories and the ‘ice cream filler’ for ice cream factories. The new building blocks all have been duplicates of existing building blocks. For example, the ‘extruder’ is based on the ‘evaporator’ building block, with three adjusted building block elements. In case of the extruder it has been the ability to have multiple in-flows and the processing time before the out-flow could start.

The new process building blocks have been composed in the same way by copying existing process building blocks and adjusting the process specific building block or by adding more parallel or simultaneous process specific building blocks.

Simulation Building Block Guideline 9: simulation building blocks and building block elements of the same family follow the same interface requirements.

All equipment building blocks part of the family of 'product containing' have the same codes for calling building block elements. At page 230 an example is described of the 'claim equipment' building block element with code 1.01. Other building block element interface codes that are used are code 1.02 for the building block element 'claim regulator' and code 1.04 for building block element 'flow in'.

Guidelines related to encapsulating internal structure of building blocks

Simulation Building Block Guideline 10: simulation building blocks hide inner working.

In simulation building blocks developed in Arena only the visual interface and the user interface are accessible for the model developer. The model developer can adjust the visual interface and set parameters in the user interface, but the model developer is not capable of viewing or adjusting the inner working of building blocks. This was a given after the selection of Arena as the generic simulation environment by Nestlé.

Simulation Building Block Guideline 11: advanced model developers have to be able to unhide the inner logic and see how the processes and attributes are implemented.

This is not possible for model developers as described in Simulation Building Block Guideline 10, due to the selection of Arena. As a workaround the process building blocks have the ability to be linked to generic model constructs of Arena via the normal connectors (Figure 8.12) or for specific connectors used in exception processes (Figure 8.13).

Mainly the second option allows experts in Arena model development to change the state of equipment via generic model constructs and thus enable the process to be continued. In the support material simple examples are provided for the use of these exception codes. The simulation models developed by the experts of Systems Navigator show advanced use to enable the production processes to continue as described by the factory analysts.

Guidelines related to providing useful services or functionality of building blocks

Simulation Building Block Guideline 12: system elements should be represented by building block elements that can be extended with custom instantiations of model constructs of a generic simulation environment.

Figure 8.13 shows how the specific process building block element of the process building block 'Transfer' is extended with extra decision logic to handle an early stoppage of the transfer. Similar capabilities are applied to the

process building blocks ‘inline mix process’, the ‘extruder process’ and ‘truck offloading process’.

Simulation Building Block Guideline 13: a building block can connect to model constructs of a generic simulation environment.

The process building blocks all contain standard Arena connectors to enable a process to be composed consisting of Nestlé process building blocks and Arena generic model constructs, for example Figure 8.12. The main reason is that the Nestlé simulation models need to contain quite some decision logic by applying reuse. Functionalities available in the generic model constructs of Arena do not need to be provided in Nestlé specific process building blocks. Therefore the simulation models of Nestlé systems commonly will be composed partially with Arena generic model constructs such as ‘Assign’, ‘Decide’, ‘Dispose’ and ‘Delay’.

Guidelines related to precisely defined interfaces for building blocks

Simulation Building Block Guideline 14: the model developer has to adjust the parameters of a simulation building block via a user interface.

This guideline has been achieved by the development of an Excel template and a Scenario Navigator start database described in section 8.4 at page 235. Both the template of Excel and the start database of Scenario Navigator have been parameterized for several simulation studies to enable factory analyst to enter the data for the experiment and view the results of the performed experiment (Excel) or experiments (Scenario Navigator).

Simulation Building Block Guideline 15: use of domain terminology in the user interface provides insight in the suitability of a building block for a certain purpose and the meaning of its parameters.

The suitability of the equipment building blocks was very clear for the Nestlé process engineers as they recognize the system element from its representation in the set of building blocks (Figure 8.8) and its visual representation in the simulation model (Figure 8.9).

The decision which process building block was to be used was more difficult for the model developers, as this required also a mentality change towards process preparation. The main example was the need to use the process building block ‘Claim equipment’ to enable the use of a piece of equipment for an order. The model developers have mainly been introduced to this concept via their training and through the error messages they received when they forgot a process step. Figure 8.18 shows an example of an error message that a model developer receives during run time because the equipment ‘Skimmed milk Silo 2’ has not been claimed to execute an order triggered by the process building block instantiated in the simulation model with the name ‘Transfer Milk’.

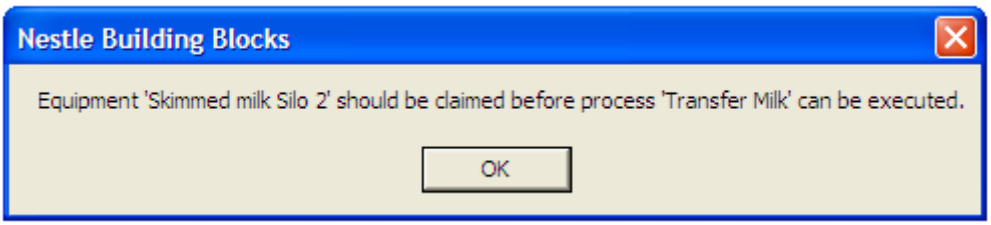


Figure 8.18: Error provided to support model developers in understanding model logic.

Simulation Building Block Guideline 16: parameters in a user interface of a simulation building block have to be checked for validity of the values.

Checks for the parameters settings are mainly carried out for the values of the equipment building blocks. Examples are the checks on storage size and the maximum rate of a product flow via a valve which both should be real numbers larger than zero. Most parameters of the process building blocks are composed expressions based on attributes or references to the recipe configuration. Therefore, the checks whether the entered data is valid cannot be performed in the parameter setting of the user interface, but has to be verified during the execution of the simulation model.

Simulation Building Block Guideline 17: parameters in a user interface of a simulation building block should have default values whenever possible.

The parameters in the equipment and process building blocks all have predefined values. The name of the building block is automatically defined, as is the identification of the valve and a default pointer to a piece of equipment. The equipment parameters like storage quantity and flow rate of the valves are also set with default values, but these are based on the initial simulation studies performed for Nestlé Nutrition. The values therefore are not the same for other facilities within the Nutrition department, and even less for facilities for ice cream, coffee or pet food.

Providing initial values mainly turned out to be useful for the small example simulation models, but for the real simulation studies, the values were invalid. Using the configuration of the Excel or Systems Navigator interface the initial parameter settings in the simulation model were overruled by values appropriate for the system to be represented.

Simulation Building Block Guideline 18: The user interface of a simulation building block should provide support for the model developer.

The texts in the user interface have been chosen to reflect as much as possible the terminology applicable within Nestlé, but in some cases it turned out to be difficult to have a common term that reflects its purpose for process engineers of Nestlé Nutrition as well as process engineers of the coffee, ice cream and pet food departments. In equipment specific for a certain

department, for example the ‘Pet food mixer’, the terminology has been agreed with the specific process engineers.

The main support for the user interface has been provided via the user manual (Valentin et al 2007). In this document each simulation building block is described with its user interface and for each parameter an explanation is given that matches the terminology of the domain. Whenever necessary, a specific explanation of a parameter is provided for multiple domains.

Simulation Building Block Guideline 19: The user interface of a simulation building block can be used by model developers to select building block elements and set their parameters.

The building blocks of the Nestlé domain specific extension do not have the opportunity to have their building block elements replaced. The equipment building blocks are dedicated to a specific equipment and therefore replacing the building block element would mean to replace the physical building block (for example replace a evaporator by a spray dryer in a Nestlé Nutrition production facility).

The process building blocks consist of simple and more advanced building blocks. If the building block elements of the advanced building blocks do not match, then the advanced building block should be replaced by a combination of simple process building blocks with Arena generic model constructs.

Simulation Building Block Guideline 20: a simulation building block has a defined interface that receives triggers, requests, entities, or events from other simulation building blocks in the simulation model and redistributes these internally.

The equipment building block uses the code determined by the process building block for the internal routing. The concept is similar as the routing mechanism used for supply chains (see chapter 6). The routing within the process building block is described in Figure 8.6, and an entity does not need a mechanism to redistribute the triggers internally in the process building block, because a sequence is used in combination with unique return labels.

Simulation Building Block Guideline 21: the interface of a simulation building block contains evaluations of the state of the trigger and the building block to determine whether the building block can handle the trigger.

Figure 8.18 is the typical example of an error message that is provided to the model developer if an equipment building block is triggered to perform an activity that does not match with its state. Important parts of the error message are the name of the process building block where the trigger initiates and the name of the equipment building block where the error occurs. This helps the model developer in identifying whether the mistake is made in the process building block and points to the wrong equipment, or that the process

for the order is incomplete and he/she should first update the state of the equipment building block.

Similar error messages as in Figure 8.18 are provided if a process building block is demanding an activity that an equipment building block cannot perform. It is very unlikely for the model developer to receive these kinds of error messages, because almost all equipment building blocks have all the building block elements and can thus handle all triggers from process building blocks. An exception is if a trigger intended for a product containing building block is sent by accident to a building block of the family of cleaning building blocks.

Simulation Building Block Guideline 22: a simulation building block contains pictures, numbers and other elements to support visualization of the state and key performance indicators during simulation run.

The visualization of an example equipment building block is provided in Figure 8.9. This screenshot shows the state of the equipment with some of its key performance indicators. The visualization of this building block is used for validating the simulation model, and for demonstrating the process to the involved stakeholders before diving into the details of the reports provided in Excel or Scenario Navigator.

8.6 Simulation studies performed

The simulation building blocks and building block elements were applied in 6 different projects at Nestlé Nutrition factories worldwide in the period of 2005-2007. Simultaneously the set of building blocks were extended to handle the simulation of processes and equipment at Nestlé factories for pet food, ice cream, coffee and soups. The extended domain specific extension for Nestlé factories was then applied in another 9 projects worldwide in 2006 and 2007. Further, over the period of three years a total of 28 process engineers were trained to use the capabilities and details of the Nestlé domain extension to analyze and develop simulations for the different Nestlé departments.

Two out of the 15 simulation projects were selected as examples. The first one is an existing Nutrition factory that is extended with new equipment. This simulation study was one of the first to be executed with the domain specific extension. The second example is a huge pet food factory in the UK that has received enormous investments over the previous years, but evaluation of the results show that the business case of these investments is not achieved. Via minor investments and improvements of the weekly production plan the expected production quantities should be achieved. The pet food factory study was one of the latest of the 15 studies using the domain specific extension release of mid-2007, which included the advanced Scenario Navigator interfacing. The other simulation studies performed for Nestlé using the domain specific extension are briefly described at the end of this section.

8.6.1 Simulation study Milk factory in India – Roadmap investment coming years

In 2005 a Nestlé factory in India was in the process of extending its group of farmers who provide fresh milk. The milk gathered from the farmers is used to make coffee milk powder. The factory management foresees that transferring a wet mixture into dry powder will be challenging for the evaporation and drying process. More fresh milk will increase the potential capacity of the factory, but fresh milk needs to be standardized and handled within a specific time to maintain freshness and quality.

The management of the factory identified a set of potential investments that might prevent a bottleneck situation during drying and evaporation, but it did not know which of these investments will provide the best result for factory performance. More important, management did not know whether investment in, for example, an extra evaporator would not cause problems in another part of the factory, for example with the storage of skimmed milk, and thus only move the bottleneck within the production system.

A simulation study needed to be performed to evaluate the effect of different investments in the factories and to provide insight into a combination of possible results. Equipment must be included and excluded, and provided at different rates and configurations in the simulation models to allow the process engineers to determine the effects of seasonal changes in milk production and of shutting down parts of the production system to allow for maintenance.

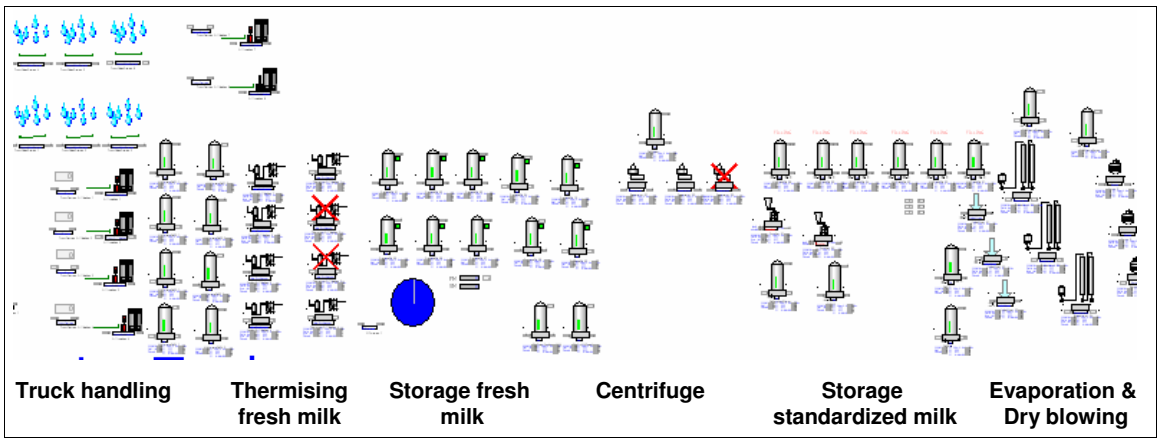


Figure 8.19: Overview equipment in simulation of Nestlé Nutrition factory

The scope of the simulation model was from the intake of fresh milk using trucks to the production of dry powder stored in totes. In between processes include temporary storage, centrifuges, mixing of product via inline batching processes and predicted bottlenecks at evaporators and/or blow drying systems. The equipment in the nutrition factory is shown in Figure 8.19, while an overview of the equipment in the part of the factory used for storage of standardized milk during the simulation run is shown in Figure 8.20.

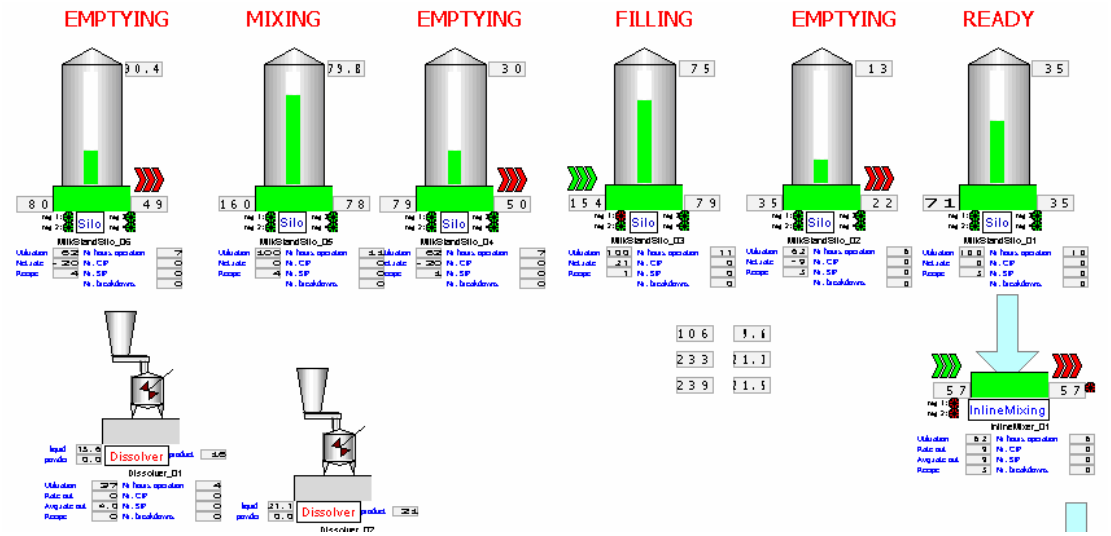


Figure 8.20: Silos for milk standardization in Nestlé Nutrition factory

The validation of the factory simulation model was performed together with the process engineers and operation managers of the Indian factory. These persons were given a detailed explanation of the different steps and details implemented in the simulation model. Data was gathered about an actual week, and the statistics of the simulation model were used to make predictions for the number of trucks required, the levels of products in tanks and silos, and to determine when activities would start and stop. These predictions were validated by the process engineers and, with some exceptions for the assumptions made with respect to trucks in the system, these predictions were correct.

The results of the simulation model are stored in an Excel interface. This interface contains all the data regarding utilization, line efficiency and production figures, similar to the data used by the factory management. Two examples of information represented in the Excel interface are shown in Figure 8.21 and Figure 8.22. The graph of the trucks in the system clearly shows the daily, repeating, process of trucks arriving in a limited time slot and waiting for the limited resources for handling the trucks. The graph of the tons of milk shows the quantity of stock at a given time for all different types of tanks, ranging from fresh milk just out of the truck to the intermediate storage for the evaporation process. The capacity in the factory and the irregularities in fresh milk delivery cause some production days where the factory is almost empty, while on other days there is up to 250 tons of milk in stock in the factory.

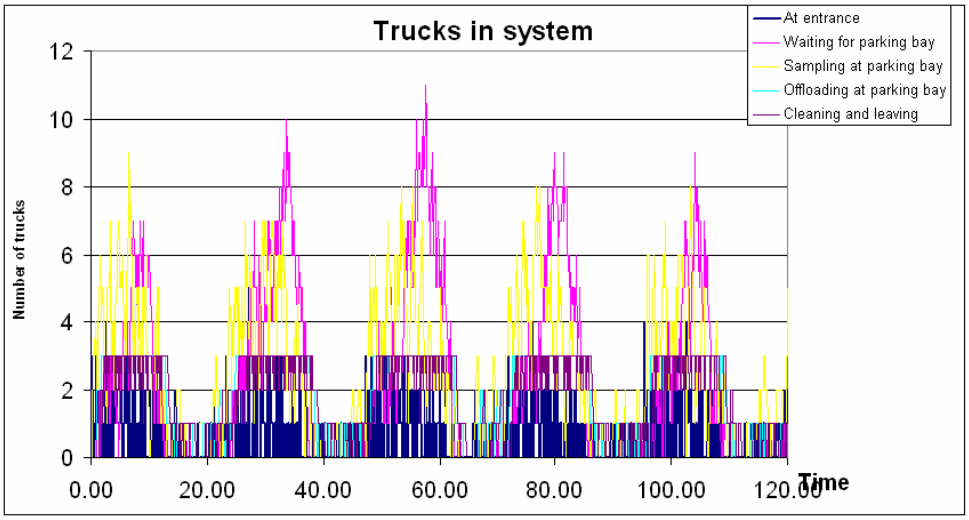


Figure 8.21: Number of trucks in factory for emptying and cleaning

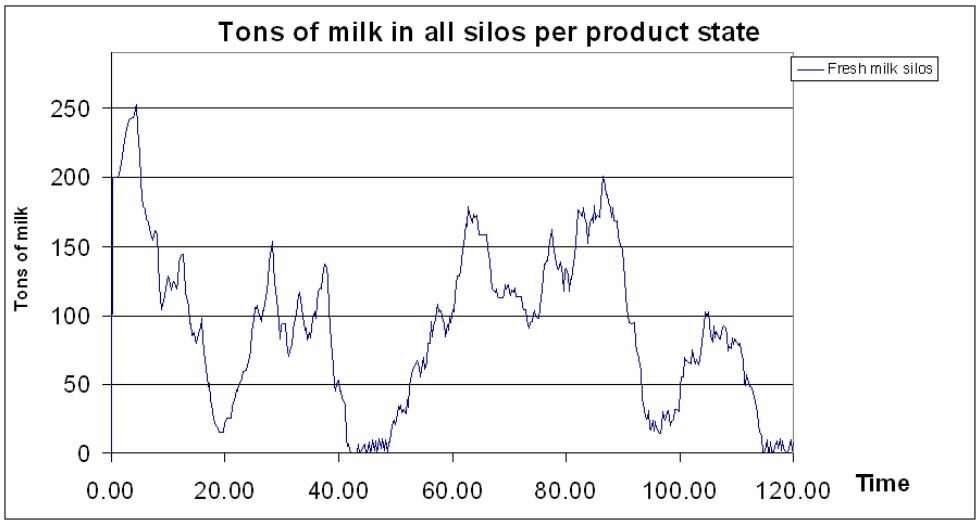


Figure 8.22: Tonnage of milk and semi products in factory

The first set of experiments was used to identify whether an increase in intake of fresh milk, i.e. more trucks coming per day, will result in a problem. The results of these experiments clearly showed that additional investments needed to be made in fresh milk storage to handle the available milk, and in increasing the production to avoid that milk stays too long in the system and needs to be thrown away.

Simulation experiments were performed to study the effects of variations to the evaporation activities. The variants could be entered simply by selecting an alternative in the Excel interface. In the simulation model this resulted in enabling and disabling all equipment that is no longer used or applicable for this particular situation. A part of the interface that refers to these settings is shown in Figure 8.23. As part of the verification process all the experiments were run once to identify the technical and conceptual correctness of the scenario. Batley (2006) claims to have performed over 150 different experiments within one week to identify the best solution and to define an

investment roadmap for the coming years. These experiments included selecting the scenario of use of spray dryers in combination with the need for storage capacity, product changes and recipe extensions in both the lean and the flush season.

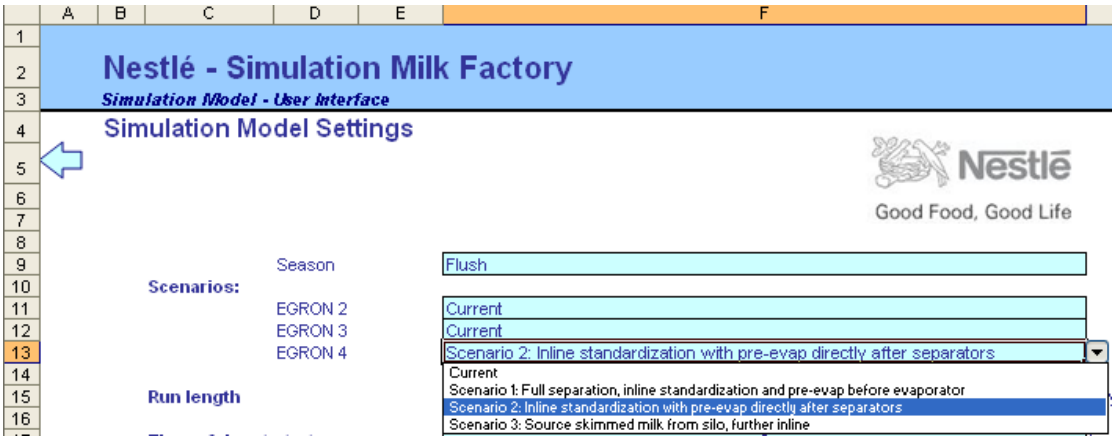


Figure 8.23: Interface to change way of evaporating per production line

8.6.2 Simulation study Pet food factory in UK – Extension and operational use

A pet food factory in the UK received over the past 25 years several investments, including several new machines to produce dry pet food that deliver products to approximately 30 filling machines. However, space of the factory is limited and the internal transport and production lines have become extremely complex. An example of the routing issues, designed by one of the analysts of the factory, who called this a ‘simple flow’, is shown in Figure 8.24.

All the new investments have increased the production capacity of the plant, but a couple of the packaging lines still work at a low efficiency rate. Logging of the factory machine data showed that this low efficiency was caused by a lack of available components which are produced by extruders, or by a temporary lack of ability to transfer a product from storage bins to the packing lines. The planners and operators had several potential solutions for these problems, these are listed below:

- The planners could try to make a better match between production of components at extruders and packing at filling lines.
- The selection of the storage location of components could take into account the need to use the filling lines efficiently.
- The mechanism used to select components and to blend them in temporary storage could be changed.
- The planning of extruders and packing lines after issues arise could be adjusted.

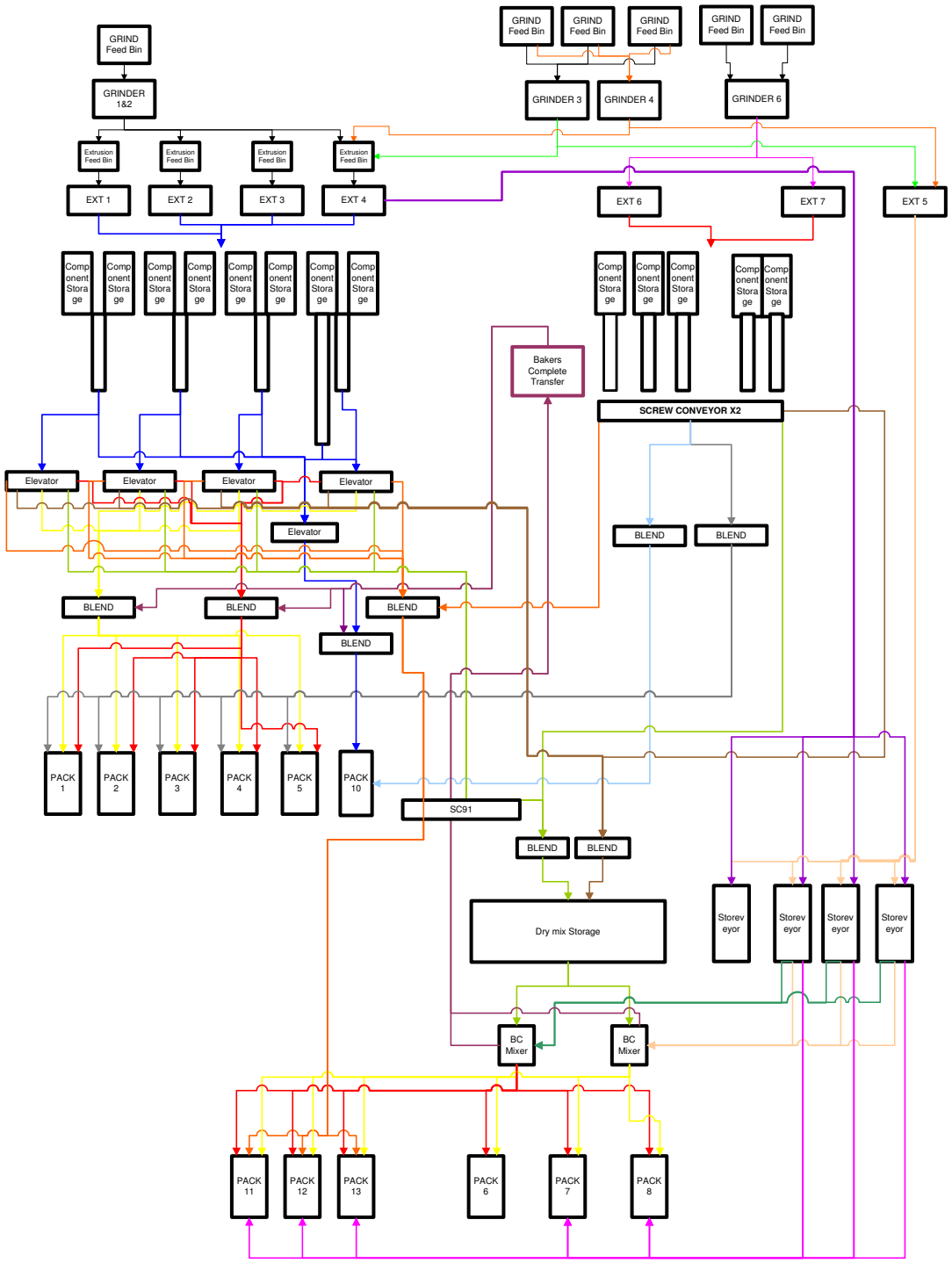


Figure 8.24: Simple process flow and routing in pet food factory

The decisions that need to be made are complex and require a lot of information about the current workload at the factory to understand and model future activities. The planners and operators currently use rules of thumb to manage their system and make operational decisions; especially when making an optimal fitting plan. This takes time, but only makes sense if it is updated with data on the latest state of the factory. The idea was to produce a simulation model of the factory, link this to the planning and use it to represent possible future situations to determine how to execute future plans. The

results can be used to improve any plans, or even better, can be included in an optimization algorithm to give a perfect plan for the coming day(s).

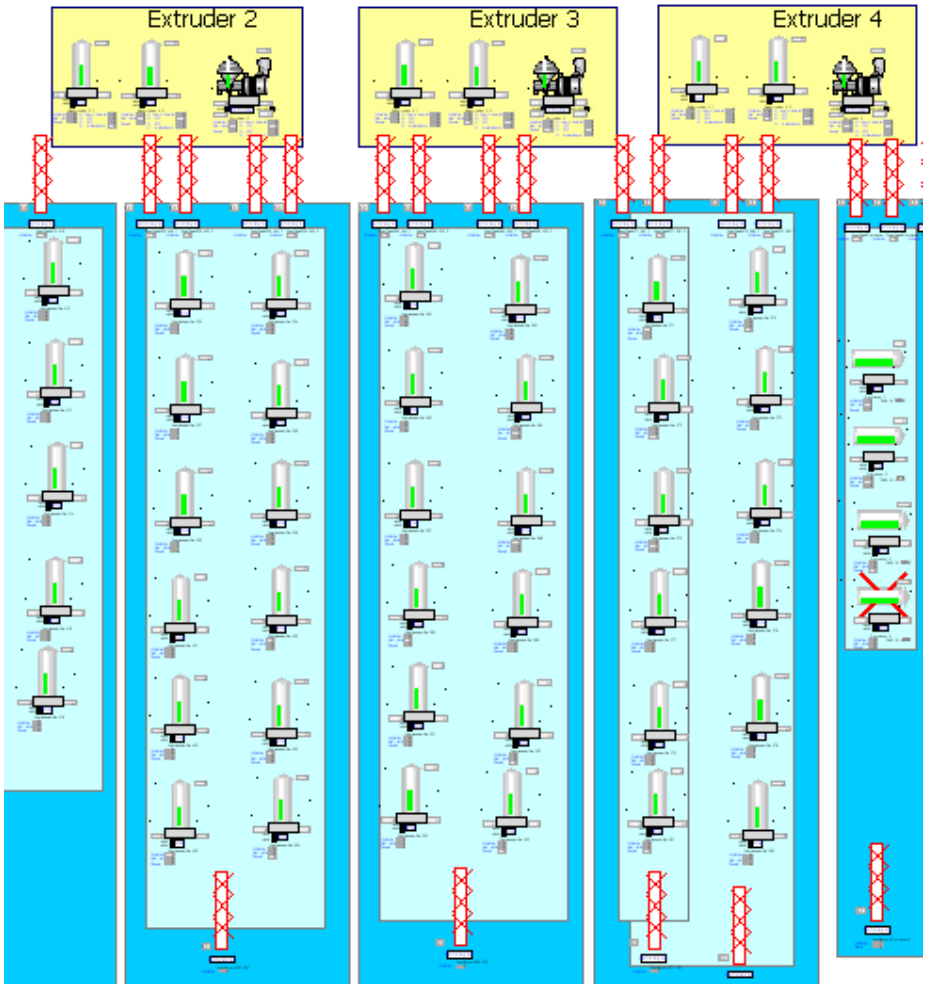


Figure 8.25: Part of storage available in pet food factory

The simulation model for the pet food factory has been instantiated using specific additional simulation building blocks. Figure 8.25 shows a part of the equipment definition of the simulation model, with at the top of the screenshot the specific pet food building blocks ‘Extruder’ to produce the components of dry pet food to be mixed in the boxes.

The verification of this simulation model was performed by evaluating, order by order, which equipment was used in which order and for what duration. Given the complex system interaction and equipment allocation these steps were done together with an analyst from the factory and this thus linked verification and validation. The validation was performed in more detail by the factory analyst via face validity checks. He performed simulation experiments using more planning rules and sets, until he found that something was not working as expected. He then described what he observed in the simulation model and what he expected and improvements were made to the simulation model, mainly the process logic of handling exceptions when a

storage silo was out of stock. This process was repeated until no differences were observed anymore.

The simulation model was used for different analyses in the factory. The simulation model has been used regularly to perform experiments with different sets of production plans and different data sets. The outcome of these regular experiments triggered the management of the factory to perform additional experiments with the simulation model. The alternatives performed with the simulation model included for example (Batley, 2006):

- A new product mix for dog food
- The effect of adding an additional package line
- The use of premixing for a specific packing line
- The advantage/disadvantage of extra storage capacity after final mixing for some products
- The advantage/disadvantage of adding additional routing from extruder to storage bins

8.6.3 Other simulation studies performed with domain specific extension

The list of the simulation studies underneath describes the factory environments where the domain specific extension was applied. At several of the facilities two simulation studies have been performed, one at the initial design and one in a later stage of the design phase when technological developments had triggered additional improvements. An example is the first case study of the new nutrition factory in the USA that has been modeled in 2004 and in 2007 with a completely different design as the initially defined UHT[5] process was not feasible.

New factory Nutrition in USA – At this factory UHT and dry powder products are produced using a total of over 60 recipes, via one production line. The challenge is when the recipe changes and new ingredients have to come online while still keeping the production line running fast enough to feed the packaging lines. The simulation model was developed by consultants at Rockwell Automation together with process engineers from Nestlé USA. The simulation model has an interface via Excel for data input. Performance indicators are imported via standard Arena reporting

New factory pet food in Eastern Europe – This new factory will start production with dry products and 4 packaging lines and the components will be imported. Two years after go-live the factory will be extended with a dedicated extruder and storage bins. The main question was whether sufficient storage bins were budgeted for given the different product mixes. The simulation model was used by process engineers from Nestlé Purina Europe to analysis, using an interface in Excel and a reduced version of the plan generator similar to the application developed for the pet food factory in the UK.

[5] UHT = Ultra-high-temperature; a specific way of processing dairy products to lengthen the expiration date.

Extension factory pet food in Eastern Europe – This factory produces dry products and will extend current production using 2 extruders with 2 new extruders, a doubled storage capacity and it will expand the number of packing lines. The main question was whether it is feasible to produce components for storage units depending on the final packing line, or whether investments are required for expensive routing and blending equipment to feed the packing lines. The experiments were performed by the process engineers of Nestlé Purina Europe, together with planners of the factory. It concluded that with different product mixes, there was no need for investment in expensive routing equipment.

Dimensioning new factory halal baby food – The production of halal baby food requires dedicated production lines, storage units and packing lines that confirm to halal requirements. A simulation model was developed to identify what size would be required for a dedicated factory for baby food. The space required would need to be enough for 10 weeks of storage before the products can be packed into boxes and cans. During this period the products need to remain in temperature controlled environments in totes (dedicated containers of approximately 2 m^3). The simulation model and interface were defined in such a way that it could be used for current evaluation of the required space, and for operational planning evaluation of the current factory. This factory has been redesigned fully when the management had decided not to support production of halal and non-halal products in one facility. The simulation experiments could all be defined within the provided Excel interface.

Extension factory pet food in Latin America – A factory producing dry products is to be extended with 2 sets of new extruders and to introduce intermediate storage for single product blends. A set of extruders allows factories to mix products directly and leads these products directly to the packing lines. The disadvantage is that if the packing line gets stuck the extruder stops. The new extruder should be a high performance extruder that can produce one type of product at high speed. The individual components are stored in bins and transferred via blending routes to the packing lines at the moment that sufficient amounts of all the types of components are in storage. The question to be answered with the simulation model was how many bins would be required for this situation. The initial design contained 20 bins, which were reduced after the simulation analysis performed by process engineers of the factory to only 14. A later design was also evaluated and this design also reduced the number of silos required.

New pro-biotic powder factory – The products made by Nutrition factories worldwide use a very small percentage of pro-biotic ingredients in their semi-finished product. This powder will be produced by a new factory that, because of the high quality requirements, is fully closed and maximally automated to avoid contamination. Using the domain specific extension, a simulation model was made by a process engineer from Nestlé Nutrition to represent the processes and filling of products into bags for worldwide transport. This

simulation model was intended to be used in a later stage for planning support and therefore contained all the planning and process delays for scheduling of processes, and for cleaning and sterilization processes. The resulting simulation model produces an evaluation of a detailed plan and can provide all required performance indicators of interest to the Nestlé factory management.

Planning improvement at a milk powder factory Asia – a milk powder factory in Asia has for the last few years been marked as a high performance factory, because they have succeeded in getting a high efficiency from their packing lines. However, because the market for their milk powder has grown, improving the efficiency of their dry blowing equipment would further improve the performance of this factory. Technically one of their dry blowers should be able to increase its average rate of production by 35%. Recently a large storage warehouse has been brought into use where the overflow from the dry blower can be stored. A simulation model was developed to be used by the planning department of the factory to analyze how changes in planning could improve the efficiency of the factory and avoid products remaining too long in the storage space. This simulation model contains an Excel interface for data entry, configuration of the plan and output representation to evaluate the success of the plan.

Planning improvement of ice cream factory in Switzerland – Nestlé produces several high quality ice cream brands in a specific factory in Switzerland. This specialized factory has, due to the broad range of products, a lot of change over times. It also faces a lot of unavailability of their filling lines due to lack of half products ready to be mixed (Valentin, 2007). A simulation model and Scenario Navigator interface have been developed with the latest domain specific extension. The Scenario Navigator interface consisted of the VB-program to define a production plan and a small optimization engine to further improve the production plan that was produced according to several parameters. The result of the combined use of the interface and the simulation model to perform the experiments enabled the factory planners to standardize the planning process and reach a 25% increase in the factory throughput during their peak season.

New production line for soup in Germany factory – The soup production is very similar to the production of milk powders. A simulation model including a planning module has been developed like the simulation development for the milk factory in Asia. The model and the interface enabled the process engineers to evaluate their initial design and on beforehand exclude some of the design alternatives even before they developed the blueprint of the factory. The factory is currently being built according to the outcome of the simulation model and the factory management has the intention to use the simulation model to support the operational planning department once the production line is fully in use.

8.7 Observations during simulation studies

The first simulation study showed the usability of the domain specific extension for Nestlé factories. The study was a straightforward investment evaluation, and the experiments helped to support decision making on future extensions. The second simulation study at the pet food factory showed that the building blocks can be used for another type of production, and for more detailed and complex processing rules. Therefore the main observation is that the domain specific extension for Nestlé production facilities is applicable across domains, extendable for new challenges and useable by process engineers with limited simulation knowledge.

The additional set of simulation studies performed by simulation experts in cooperation with Nestlé process engineers showed that the domain specific simulation extension could be used by different model developers, and that the resulting simulation models made for different types of factories for different types of analyses could be used by Nestlé process engineers. Interaction with Scenario Navigator made it possible to link the system to SAP Globe and to determine operational state information for the factory. The second main observation is thus that the simulation models not only are suited for initial investment decisions, but also can support operational planning with optimized production schedules.

8.7.1 Observations regarding design approach and implementation

The design of the domain specific extension was a step wise approach whereby the full scope was not worked out in detail on beforehand. The focus in the early days of the project was on getting the first set of building blocks defined and working. The result was that with every new simulation model one or more adjustments were made to the domain specific extension to improve the building blocks, extend the scope, or fix problems. It was positive that the structure of the domain specific extension allowed all these changes. It was also positive that the management of Nestlé was supportive of the approach and the project as a whole.

It was less positive that changes to the domain specific extension sometimes had implications to past simulation models, where it was not always possible to change to the newest version. The applied workaround was the use of different versions of the domain specific extension for Nestlé. At some moment in time three different sets of simulation building blocks were available to support three different simulation studies. Once these different simulation studies were finished, the latest domain specific extension was used for further development.

8.7.2 Observations regarding additional tools

The Excel interface and the Scenario Navigator database both helped the analysts in entering their data and retrieve the simulation results. The extension with the plan generator to develop a production schedule for the

simulation model also proved to be successful. A positive fact about the interfaces was that they were all based on the same template, thus for all simulation models the interfaces looked similar, which enabled recognition, yet they were focusing on specific needs of the analyst team of a certain simulation project.

8.7.3 Observations regarding provided support

The training for learning to use the domain specific extension only worked in combination with a short Arena introduction. The Nestlé process engineers picked up the basics rather quickly and succeeded in carrying out their assignments successfully, but the step between the simple assignments and the free format end assignment was rather large. The step from the free format assignment to the development of a simulation model of a complete Nestlé production facility turned out to be even larger. The training did provide sufficient support, however, so that the Nestlé process engineers understood the concept, the background of the building blocks, and the reason why simulation experts asked certain questions during the modeling process.

The user manual and the small simulation models have therefore hardly been used by the process engineers, but mainly for validation purposes. Nevertheless, the user manual provided support in explaining the importance of parameters and to gain confidence that the simulation building blocks were not just a one-off project.

8.7.4 Observations regarding applying the guidelines for simulation building blocks

The observations for the capability of the domain specific extension and to result in genuine simulation building blocks is structured via the characteristics of a building block as defined by Verbraeck et al (2002) in the table at the next page.

Table 8.1: Characteristics of building block in Nestlé case study

Self-Contained	Positive	use of building block elements in several building blocks & data made available via the Arena expression builder.
Interoperable	Positive	separation in infrastructure and processes & simple equipment building blocks in combination with flexible process building blocks.
Reusable	Positive	applicability in several Nestlé domains.
Replaceable	Positive	process building block flexibility & ability to implement complex processes with process building blocks and generic model constructs.
	To be improved	hierarchy structure in equipment building blocks.
Encapsulating its internal structure	Positive	hidden inner-working & good capability to extend with generic model constructs.
Providing useful services or functionality	Positive	equipment could provide all required functionalities via flexible processes & processes extendable by new process building block elements.
Precisely defined interfaces	Positive	user interfaces for parameters also available via additional tool & visualization representation includes key performance indicators, yet is recognizable by process engineers & technical structure for information exchange good organized via coding.

8.8 Overview observations

8.8.1 Observed benefits

The simulation studies that were performed in the domain of Nestlé's production facilities were confirmation of the benefits that were noted in chapter 2 and 3. Overall, the simulation studies were performed correctly and the problem owners were satisfied with the output, resulting in new assignments and a scope growing from dairy to ice cream, coffee and pet food.

The only remark to be made to the achieved benefits is for benefit 4.3 'model development by simulation novices'. During the training of the domain specific extension the process engineers of Nestlé were capable to develop small simulation models, but the development of a simulation model of a complete Nestlé production facility was too much to ask. The main reason was that it required quite some modeling expertise to translate the production processes into a model and the use of the domain specific extension also required the use of generic Arena model constructs.

Table 8.2: Summary of benefits observed in case study Supply Chain

Process step **Expected benefits as mentioned in chapter 2 and 3**	**Observation Nestlé**
Activity 1: Problem description & define conceptual model	
Benefit 1.1: conceptualize system elements with model constructs in mind	Yes
Activity 2: Select model constructs	
Benefit 2.1: no translation between system elements and model constructs	Yes
Benefit 2.2: compose model constructs from developed domain specific model constructs to represent system elements	Yes
Benefit 2.3: easy selection of model construct thanks to structure of domain specific extension	Yes
Activity 3: Data collection	
Benefit 3.1: collection of predefined input data	Yes
Activity 4: Instantiate simulation model for original system	
Benefit 4.1: less model constructs used	Yes
Benefit 4.2: model development faster and easier	Yes
Benefit 4.3: model development by simulation novices	Partly
Activity 5: Verify and validate simulation model for original system	
Benefit 5.1: no more detailed testing	Yes
Benefit 5.2: easily gathering validation data	Yes
Benefit 5.3: structured and standardized performance indicators	Yes
Benefit 5.4: semi-automatic reporting of performance indicators	Yes
Benefit 5.5: observe animation at different levels of the composition: high level and at individual model construct	Yes
Activity 6: Analyze output of simulation model	
Benefit 6.1: structured and standardized performance indicators	Yes
Benefit 6.2: semi-automatic reporting of performance indicators	Yes
Activity 7: Define solution for analyzed outcome	
Benefit 7.1: model developers are triggered to find new solutions by parameters	Yes
Activity 8: Instantiate simulation model for identified solution	
Benefit 8.1: easy adjustment of model thanks to user interfaces of model constructs	Yes
Benefit 8.2: easy adjustment of model thanks to replacement of model constructs	Yes
Benefit 8.3: easy visualization thanks to incorporation of visualization in model constructs	Yes
Benefit 8.4: composition of new model constructs enabled new solutions to be evaluated	Yes

8.8.2 Observed risks

The simulation studies for Nestlé contained all complexity and time pressure that model developers and problem owners are used to. Even though, the observations show that almost all the risks identified in chapter 2 and 3 have been mitigated by the guidelines and design approach of chapter 5. Similar to in the tables in chapter 3, potential risks that we did not observe during the execution of any of the simulation studies within this case study ("No" in the table) probably did not occur and thus the potential risk has been mitigated by the way the domain specific extension was designed, structured and used.

Table 8.3: Summary of risks observed in case studies

Process step **Potential risks as mentioned in chapter 2 and 3**	**Observation Nestlé**
Activity 1: Problem description & define conceptual model	
Risk 1.1: scope of model developer is limited by model constructs	No
Activity 2: Select model constructs	
Risk 2.1: lack of trust results in no motivation to use domain specific extension	No
Risk 2.2: lack of insight in model constructs results in ignore domain specific extension	No
Risk 2.3: use of model constructs that are not suited for representation of system elements	No
Risk 2.4: system elements can not be represented by model constructs	No
Risk 2.5: compose model constructs from developed domain specific model constructs only applied for infrastructure system elements	No
Risk 2.6: model developers can adjust internal logic of model constructs	No
Activity 3: Data collection	
No risks defined in chapter 2 or 3	
Activity 4: Instantiate simulation model for original system	
Risk 4.1: model developers do not understand model construct	No
Risk 4.2: model developers do not know how to parameterize model construct	No
Risk 4.3: difficult to compose simulation model, because model constructs are not available	Partly
Risk 4.4: difficult to compose simulation model by person other than developer(s) domain specific extension	Rarely

(continued at next page)

Activity 5: Verify and validate simulation model for original system	
Risk 5.1: mistakes of model developer are hard to overcome	No
Risk 5.2: model developers know something is wrong, but cannot identify what to do about it	Partly
Activity 6: Analyze outcome of simulation model	
Risk 6.1: model constructs do not provide performance indicators problem owner desired	Partly
Activity 7: Define solution for analyzed outcome	
Risk 7.1: model developers are triggered to find new solutions by parameters	No
Risk 7.2: model developers are limited by parameters and model constructs	No
Activity 8: Instantiate simulation model for identified solution	
Risk 8.1: solution is identified that cannot be represented by model constructs	No
Risk 8.2: adjustments of model constructs required to represent solution are time consuming	No
Risk 8.3: replacement of model constructs causes errors in model constructs that were linked or connected.	No

The simulation models for Nestlé encountered several risks regarding the availability of model constructs and performance indicators, i.e. risks with 'Partly'. This reflects that the initial scope of the domain specific extension was on purpose not sufficient for all production facilities of Nestlé. The risk was thus observed that for example the pet food systems could not be represented. The solution was that the sets of simulation building blocks of the domain specific extension were extended with simulation building blocks for petfood. After the extension the domain specific extension could be used to represent the simulation model for the new sub-domain. The same applied to the required extensions for ice cream and coffee (Valentin et al, 2007).

Some of the simulation studies of Nestlé production facilities also observed the risk 'difficult to compose simulation model by person other than developer(s) domain specific extension'. The simulation models have been developed mainly by simulation experts, either the developers of the domain specific extension or simulation experts closely contact to the developers of the domain specific extension. The reason was not that it was not possible to develop the simulation models without detailed knowledge of the simulation building blocks, but the process configuration of a Nestlé production facility contained quite some exceptions that had to be developed with a mixture of Nestlé process building blocks and Arena generic model constructs. The knowledge that simulation experts had about the inner working of the process building blocks made the development of the simulation model easier. In the

training attention was paid to this point, but 5 days was insufficient to teach the full scope of capabilities of Arena in combination with the Nestlé process building blocks (Valentin, 2007).

9 Epilogue

9.1 Introduction

The completion of the three case studies described in chapter 6, 7 and 8 provides us with observations for the theory proposed in chapter 5 for domain specific extensions. It was proposed to develop domain specific extensions using four elements: building blocks, design approach, additional tools and support and documentation. At the end of the case study chapters we already posed several remarks and conclusions regarding the observed benefits and encountered risks. In section 9.2 we combine these observations of the three case studies (Supply Chains, Container Terminals and Nestlé Production Facilities)[6]. In section 9.3 we provide a generalization of the observations in relation to the requirements for domain specific extensions stated in section 5.2. In section 9.4 we translate the observations and generalizations of the case studies to answer the research questions identified in chapter 1. Finally in section 9.5 we look forward to points of interest for further investigation so we can further improve the theory, leading to continuous improvement of domain specific extensions in simulation studies.

9.2 Combined observations of case studies

9.2.1 Overall conclusions regarding benefits

In chapter 2 we introduced a set of benefits for the use of domain specific extensions (section 2.4; table 3.6). After the execution of the case studies in chapter 3 we extended the initial list of benefits with table 3.7. We listed whether we observed these benefits in the individual chapters of the testing case studies (sections 6.8.1, 7.8.1 & 8.8.1). Table 9.1 provides the observations of the three testing case studies together. In this table we clearly see that all benefits originating from literature (introduced in chapter 2) and possible benefits observed in the initial case studies (identified in chapter 3) have been encountered in all three testing cases, with the remark that some of the benefits for the case study container terminal have not been achieved on purpose.

We therefore conclude that the theories introduced in chapter 5 have enabled the testing case studies to be successful and that the theories enable to achieve the benefits expected to be achieved with the use of domain specific extensions in all activities of discrete event simulation studies.

[6] We performed more simulation studies besides the three case studies described in chapter 6, 7 and 8 over the period of the research project. These case studies are briefly described in Appendix I. Whenever applicable a reference to these cases is included and the observations for these case studies are used as well.

Table 9.1: Summary of benefits observed in case studies

Process step **Expected advantages as mentioned in chapter 2 and 3**	**Supply chains**	**Container**	**Nestlé**
Activity 1: Problem description & define conceptual model			
Benefit 1.1: conceptualize system elements with model constructs in mind	Yes	Yes	Yes
Activity 2: Select model constructs			
Benefit 2.1: no translation between system elements and model constructs	Yes	Yes	Yes
Benefit 2.2: compose model constructs from developed domain specific model constructs to represent system elements	Yes	Yes	Yes
Benefit 2.3: easy selection of model construct thanks to structure of domain specific extension	Yes	Yes	Yes
Activity 3: Data collection			
Benefit 3.1: collection of predefined input data	Yes	Yes	Yes
Activity 4: Instantiate simulation model for original system			
Benefit 4.1: less model constructs used	Yes	Yes	Yes
Benefit 4.2: model development faster and easier	Yes	Yes !	Yes
Benefit 4.3: model development by simulation novices	Yes	Yes !	Partly
Activity 5: Verify and validate simulation model for original system			
Benefit 5.1: no more detailed testing	Yes	Yes	Yes
Benefit 5.2: easily gathering validation data	Yes	Partly	Yes
Benefit 5.3: structured and standardized performance indicators	Yes	Yes	Yes
Benefit 5.4: semi-automatic reporting of performance indicators	Yes	Yes !	Yes
Benefit 5.5: observe animation at different levels of the composition: high level and at individual model construct	Yes	Partly	Yes
Activity 6: Analyze outcome of simulation model			
Benefit 6.1: structured and standardized performance indicators	Yes	Yes	Yes
Benefit 6.2: semi-automatic reporting of performance indicators	Yes	Yes !	Yes
Activity 7: Define solution for analyzed outcome			
Benefit 7.1: model developers are triggered to find new solutions by parameters	Yes	Partly	Yes
Activity 8: Instantiate simulation model for identified solution			
Benefit 8.1: easy adjustment of model thanks to user interfaces of model constructs	Yes	Yes	Yes
Benefit 8.2: easy adjustment of model thanks to replacement of model constructs	Yes	Yes	Yes
Benefit 8.3: easy visualization thanks to incorporation of visualization in model constructs	Yes	Partly	Yes
Benefit 8.4: composition of new model constructs enabled new solutions to be evaluated	Yes	No	Yes

9.2.2 Overall conclusions regarding risks

In Table 9.2 we put the risks together from the different testing case studies. We can clearly see in our observations of these testing case studies that all risks have been mitigated in these simulation studies. Some of the risks mentioned in the table have been encountered during the execution of the case studies. These risks have to do with the decision to extend or not to

extend the domain specific extension with new model constructs, instead of making a complete domain specific extension at the start of the first simulation study.

The encountered risks for the testing case study of supply chains were caused by scope of the implementation which varied for the used generic simulation environment. The proof of concepts developed in the generic simulation environments of eM-Plant and D-SOL only contained the simulation building blocks which where necessary to demonstrate the concept. The teaching cases performed with the domain specific extension developed in Arena contained more simulation building blocks and building block elements to represent alternative system elements. Still, the scope was to support the teaching cases and the implemented scope did not contain all supply chain concepts that Corver (2000) defined.

The case study for container terminals aimed at providing a black box with limited features for designing and evaluating a container terminal in 15 minutes. Given the time limit, there was no room for extensions or changes to simulation building blocks. Whenever the game participants encountered a limit of the system, they decided to park the option for further research and therefore the missing scope was no limitation to the progress of the game.

For the Nestlé case, it was decided with the management of Nestlé to develop the domain specific extension stepwise by adding new simulation building blocks with every new simulation study executed. Therefore, some of the risks (e.g., “1.2: system elements cannot be represented by model constructs”) have been encountered in some of the simulation studies for Nestlé, but these risks have been mitigated in follow-up studies to update the domain specific extension.

Table 9.2: Summary of risks observed in testing case studies

Process step **Potential risks as mentioned in chapter 2 and 3**	**Supply chains**	**Container**	**Nestlé**
Activity 1: Problem description & define conceptual model			
Risk 1.1: scope of model developer is limited by model constructs	No	Partly	No
Activity 2: Select model constructs			
Risk 2.1: lack of trust results in no motivation to use domain specific extension	No	No	No
Risk 2.2: lack of insight in model constructs results in ignore domain specific extension	No	No	No
Risk 2.3: use of model constructs that are not suited for representation of system elements	No	No	No
Risk 2.4: system elements can not be represented by model constructs	No	No	No
Risk 2.5: compose model constructs from developed domain specific model constructs only applied for infrastructure system elements	No	No	No
Risk 2.6: model developers can adjust internal logic of model constructs	No	No	No

(continued at next page)

Activity 3: Data collection			
No risks defined in chapter 2 or 3			
Activity 4: Instantiate simulation model for original system			
Risk 4.1: model developers do not understand model construct	No	No	No
Risk 4.2: model developers do not know how to parameterize model construct	No	No	No
Risk 4.3: difficult to compose simulation model, because model constructs are not available	No	Partly	Partly
Risk 4.4: difficult to compose simulation model by person other than developer(s) domain specific extension	No	No	Rarely
Activity 5: Verify and validate simulation model for original system			
Risk 5.1: mistakes of model developer are hard to overcome	No	No	No
Risk 5.2: model developers know something is wrong, but cannot identify what to do about it	No	No	Partly
Activity 6: Analyze outcome of simulation model			
Risk 6.1: model constructs do not provide performance indicators problem owner desired	No	No	Partly
Activity 7: Define solution for analyzed outcome			
Risk 7.1: model developers are triggered to find new solutions by parameters	No	No	No
Risk 7.2: model developers are limited by parameters and model constructs	Partly	Partly	No
Activity 8: Instantiate simulation model for identified solution			
Risk 8.1: solution is identified that can not be represented by model constructs	No	Partly	No
Risk 8.2: adjustments of model constructs required to represent solution are time consuming	No	No	No
Risk 8.3: replacement of model constructs causes errors in model constructs that were linked or connected.	Partly	No	No

9.3 Matching of requirements for domain specific extensions

In chapter 2 and 3 we already concluded that the use of domain specific extensions enables some of the activities for a simulation study to be performed faster, easier, less detailed etcetera resulting in the benefits listed in Table 3.6 and Table 3.7. We kept the activities in the simulation study the same, even though the actual actions within the activity in some cases have been reduced. Instead we focused in chapter 5 on providing concepts, guidelines, approaches and suggestions for tools to avoid the risks we observed. This resulted in the 9 requirements in section 5.2. In this section we explain how we satisfied the requirements and demonstrated this in the case studies of chapter 6, 7 and 8.

9.3.1 Matching the concept of simulation building blocks to the requirements

In section 5.4 we described the generic concept of building blocks (Verbraeck et al, 2002) and the translation of this concept to simulation building blocks with simulation building block elements. The 22 guidelines define what is expected of a simulation building block and what points a simulation model developer should pay attention to.

The main point of these guidelines is the introduction of the use of building block elements underlying the simulation building blocks. Building block elements enable that building blocks are self-contained, interoperable, reusable, replaceable, encapsulate their internal structure and provide a useful service via their interfaces. We borrowed concepts from software engineering such as product line engineering and interfacing to structure simulation building blocks and building block elements. We enriched this further with the capabilities of generic simulation environments regarding user interfacing and visualization. In addition, we reused the experience we gathered in the case studies of chapter 3, AGVs and passengers at airports, regarding infrastructure versus control mechanisms, building block families, and the use of terminology of the problem owner.

Table 9.3 shows how these concepts and guidelines worked out to match the requirements to domain specific extensions. The contribution of the requirement can be observed in all case studies. A typical example from one of the case studies of chapter 6, 7 or 8 is provided in the third column of Table 9.3.

Table 9.3: Contribution simulation building blocks to domain specific extensions

Requirements for domain specific extensions	**Contribution simulation building blocks**	**Example from case study**
Requirement 1: DSE should show added value for model developers compared to use of model constructs of generic simulation environments	Model developers are supported by simulation building blocks as they are ready to use and recognizable model constructs.	Nestlé: simulation studies defaulted to being executed using DSE
Requirement 2: Use of model constructs of DSE should be clear and well defined so model developers know when and how to use the model constructs	Simulation building blocks in terminology of problem owner sorted in a family are easy to match to need of model developer, user interface provides further suggestions and support.	Supply Chains: concept of modeling a supply chain translated to building block elements in structured families

(continued at next page)

Requirements for domain specific extensions	**Contribution simulation building blocks**	**Example from case study**
Requirement 3: System elements that seem to be exceptional for the domain represented by the DSE should not become model constructs	Building block elements enable definition of different simulation building blocks, and special interfaces of the simulation building block allow for using model constructs of generic simulation environment to incorporate specific logic and control whenever necessary in simulation model	Nestlé: process description in simulation models composed of simulation building blocks and model constructs using the full strength and flexibility of the generic simulation environment
Requirement 4: The infrastructure and physical elements should be represented by model constructs separated from the model constructs for control or management	Apply specialization of simulation building blocks in representation of physical elements and process-like description, further supported by family structure of building blocks	Supply Chains: Logical and physical actors with different building block elements each represent part of the supply chain process
Requirement 5: Internal logic of model constructs of DSE should be closed or accessible depending on type of model developer	User interfaces provide ability to hide or show the inner logic.	Supply Chain implementation in eM-Plant: model developer can override logic in building block elements
Requirement 6: Model constructs should be understandable for model developers	Apply terminology used in the domain and by problem owner.	Nestlé: machines have different names for the icecream and petfood factory
Requirement 7: DSE should be an extendible set of model constructs	Functionalities, level of detail and scope of simulation building block can be adjusted by replacing building block element with a new building block element that better matches the requirement of the model developer	Supply Chains: different variants of order generation building block element, to be extended in many different ways

Requirements for domain specific extensions	Contribution simulation building blocks	Example from case study
Requirement 8: Behavior of model construct should be understandable and verifiable	Simulation building block and building block element have visualization (animation and performance indicators) and parameters which can be observed in the model	Supply Chains: large set of graphs and plots to show the status of the building block, which change when the model runs.
Requirement 9: Model constructs should be individually parameterizable	Simulation building blocks have a dedicated user interface for their parameters	Container Terminals: each storage can have different stacking height for containers

In the development of the simulation building blocks and the building block elements according to the guidelines for the 11 different domains described in appendix I many similar choices have been made. The same types of building blocks were created, and also the same structure in parameter settings and visualization was observed.

9.3.2 Additional tools for domain specific extensions matched to requirements

The simulation building blocks were an improvement in user friendliness to the model developer compared to the model constructs. Additional tools have been introduced to support the model developer in time consuming activities that can be automated. The additional tools focus on repetitive activities for model construction, parameterization of simulation building blocks and gathering output data into a useful report with performance indicators.

Especially the case study of the management game 'Container Adrift' proved that additional tools could support the model developer and model analyst much further. The additional tools have been applied to supply chains and Nestlé and some of the domain specific extensions in appendix I, but their effect has been less explicit. Nevertheless, the simulation studies at for example the Nestlé factories would have been completely different without the Excel interface or the 'plan generator' application in combination with Scenario Navigator, as these additional tools supported to keep model parameterization together with output results of a specific simulation experiment.

Only a couple of the requirements of section 5.2 apply to additional tools. Therefore Table 9.4 only shows the applicable requirements and how the use of additional tools support the requirement. An example is provided in the third column based on the case study of the container terminal, chapter 7, as this case study focused around the additional tools.

Table 9.4: Contribution additional tools to domain specific extensions

Requirements for domain specific extensions	Contribution additional tools	Example from case study Container Terminals
Requirement 1: DSE should show added value for model developers compared to use of model constructs of generic simulation environments	Compose the simulation model automatically, set parameters and provide a specific report including all output data of the simulation model without user intervention	Simulation model development, execution and evaluation of performance indicators could be performed by non-experts within 15 minutes.
Requirement 2: Use of model constructs of DSE should be clear and well defined so model developers know when and how to use the model constructs	Interface for parameterization and output report via Excel provides extra information and support material to model developer	Customized Excel sheet with performance indicators contained additional controls to present model developers with potential risks in their design
Requirement 9: Model constructs should be individually parameterizable	Parameterization via additional tools such as database or Excel supports quick adjustment of all model constructs with new individual values	Company information with all details could be easily included by number of clicks in database part of the container modeling solution.

9.3.3 Support and documentation for domain specific extension matched to requirements

Support and documentation focuses on enabling the model developer to better understand the capabilities of the domain specific extension and to be able to apply the domain specific extension in a simulation study. The support is provided by example models, small models to demonstrate specific situations, by large models as show cases, and by training material. The documentation focuses on what elements of a simulation building block or building block element should be defined. This is to support the future users of the domain specific extension, but also the future developers who will add extra building block elements and simulation building blocks to the domain specific extension.

In the three case studies the support and documentation was provided in different ways. Training in the management game ‘Container Adrift’ was about the use of the visualization-simulation tool and not about the use of the simulation building blocks. Training for Nestlé was to prepare the process engineers for the questions from the simulation experts who were developing

the simulation models, and training of the students at the R.H. Smith Business School of the University of Maryland for the supply chain teaching cases was mainly about interpreting the results of the performed simulation models. In the end the documentation as described in chapter 5 according to the structure based on Heisel and Souquières (2004) was provided to the simulation model developers, but they usually decided not to use this material.

Nevertheless, the stakeholders in the simulation studies, e.g. the process engineers of Nestlé advising the factory management, would not have been so trusting if the extensive documentation would not have been present. They would have had doubts about the quality and the maturity of the domain specific extension. The question is whether the extensive documentation was necessary, in other words, would the example simulation models and testimonies of simulation studies with the domain specific extension have been sufficient?

Support and documentation is an important ingredient to ensure that the model developer understands and trusts the domain specific extension, and therefore all requirements of section 5.2 have an element of support and documentation, but in the case studies these topics have been covered by dedicated and customized training with hands-on contributions by one of the developers of the domain specific extension. Therefore Table 9.5 contains only the requirements where additional contributions were addressed, supported by an example of one of the three case studies.

Table 9.5: Contribution documentation and support to domain specific extensions

Requirements for domain specific extensions	**Contribution documentation and support**	**Example from case study**
Requirement 1: DSE should show added value for model developers compared to use of model constructs of generic simulation environments	Success stories of performed simulation projects with the domain specific extension.	Nestlé: via the milk factories the idea of using simulation spread to petfood, coffee and icecream
Requirement 6: Model constructs should be understandable for model developers	Small simulation models that demonstrate one or two specific simulation building blocks or building block elements.	Nestlé: Over 100 different small simulation models have been developed, each containing less than 10 simulation building blocks.

Requirements for domain specific extensions	Contribution documentation and support	Example from case study
Requirement 8: Behavior of model construct should be understandable and verifiable	Training material contains assignments with verified output and provided solution models.	Supply Chains: specific assignments to understand the parameters and train interpretation of results.

9.3.4 Design approach for improved domain specific extensions matched to requirements

The process to develop a domain specific extension was described in section 5.7 using Figure 5.18 and 5.19. The first figure provided an overview from initial design until the final handover to future model developers. The second figure provides details for the conceptualization and specification of the design of the simulation building blocks and the building block elements as part of the domain specific extension.

The main observation from the three case studies, supported by the cases in Appendix I, is that the structured approach of specification provides a good preparation for the implementation, either directly for the initial domain specific extension, or later for updates and extensions to the domain specific extension using the flexibility of composing building blocks out of building block elements. Also we learned that the design approach is suited for different implementations and that the design approach is thus independent of the future generic simulation environment that is going to be used to realize the domain specific extension.

Key element in the specification phase is the use of lists to verify that the designs can be realized and that the decision choices do not hinder the intended use of the domain specific extension too much. The case studies of supply chains and Nestlé demonstrate that simulation models can be used in different situations, i.e. real time gaming and production planning. No attention was paid in the design of simulation building blocks to potential usage for real-time situations or to create a simulation model of the system from a predefined state. As these types of experiments were originally left out of scope of the domain specific extension, the realization of the new type of experiments was a bigger challenge, but still achievable by introducing new building block elements.

The design approach supports the developers to construct a domain specific extension that follows the simulation building block guidelines, provides additional tools, and that is well supported and documented. All of this is done in close cooperation with problem owners to ensure they recognize the building blocks and terminology. The requirements of section 5.2 aimed at the products to be delivered, not to the design approach that supports smooth delivery. Table 9.6 shows the requirements that resulted in

contributions for the design approach, with an example of one of the case studies of chapter 6 (Supply Chain) or 8 (Nestlé).

Table 9.6: Contribution design approach to domain specific extensions

Requirements for domain specific extensions	Contribution design approach	Example from case study
Requirement 1: DSE should show added value for model developers compared to use of model constructs of generic simulation environments	Conceptualization and specification of the domain specific extension is extremely important. The design approach pays attentions to the future experimentation and objective of simulation studies and includes checks in each design decision	Nestlé: A list of experiments and variants has been defined in a workshop with over 20 functional experts regarding their expectations of the simulation studies
Requirement 3: System elements that seem to be exceptional for the domain represented by the DSE should not become model constructs	Not everything can be implemented before the first simulation study is started. The building block elements should be kept open and by setting priorities in the design approach further implementation should be scheduled.	Nestlé: Initially developed for milk products. A proof of concept has been performed for the petfood division, followed by an implementation plan for several years to develop petfood specific building blocks.
Requirement 6: Model constructs should be understandable for model developers	Discuss extensively with problem owners in domain to ensure terminology and problems are well covered in the simulation building blocks	Supply Chains: A study has been performed by Corver to define concepts for modeling the supply chains based on literature study and experts at RH Smith business school.

9.3.5 Match requirements and theory for domain specific extensions

Table 9.7 shows how the requirements are satisfied by the new theory described in chapter 5 and the table identifies whether this theory solution for the requirements can be observed in the case studies of chapters 6, 7 and 8. The solution directions (simulation building blocks; design approach; support simulation study execution; documentation) all contribute to match the defined requirements. The columns at the right hand side show whether the requirement of a domain specific extension has been successfully matched for

the extension developed in the case study. Some of the requirements have not been satisfied on purpose, due to the scope of the case study. This has been mentioned in the applicable situations. Overall our observation is that we covered all other requirements for the domain specific extensions.

Table 9.7: Representation of requirements in theory and case studies

Requirements for domain specific extensions	Observations		
	Supply Chain	**Container terminal**	**Nestlé**
Requirement 1: DSE should show added value for model developers compared to use of model constructs of generic simulation environments	Yes	Yes	Yes
Requirement 2: Use of model constructs of DSE should be clear and well defined so model developers know when and how to use the model constructs	Yes	Yes	Yes
Requirement 3: System elements that seem to be exceptional for the domain represented by the DSE should not become model constructs	Yes	Exceptional system elements were not part of the game.	Yes !
Requirement 4: The infrastructure and physical elements should be represented by model constructs separated from the model constructs for control or management	Yes !	Yes, especially physical and logical split	Yes !
Requirement 5: Internal logic of model constructs of DSE should be closed or accessible depending on type of model developer	Yes, especially eM-Plant implement ation	Yes, for inner working and structure parameters.	Yes
Requirement 6: Model constructs should be understandable for model developers	Yes	Yes	Yes
Requirement 7: DSE should be an extendible set of model constructs	Yes	Yes	Yes
Requirement 8: Behavior of model construct should be understandable and verifiable	Yes	Yes	Yes
Requirement 9: Model constructs should be individually parameterizable	Yes	Yes !	Yes !

9.4 Answers to the research questions

In chapter 1 we introduced three research sub-questions to clarify how domain specific extensions can help a model developer to perform simulation studies without the simulation studies becoming ineffective. Based on Robinson and Pidd (1998) our research questions focused on: handle unlimited modeling freedom, support model developers to cover multiple fields of expertise, and resolve language mismatches. Chapter 5 describes four elements as an addition to the existing theory for domain specific extensions:

1) the concept of simulation building blocks (section 5.4)
2) the use of additional tools (section 5.5)
3) the importance of support and documentation (section 5.6)
4) the design approach (section 5.7)

These four elements have been applied in the case studies that have been executed as part of this research, which were described in detail in chapter 6, 7 and 8 and at a high level in appendix 1. Each of the three case studies that was described in detail focuses on one of the points of Robinson and Pidd regarding ineffective simulation studies. “Unlimited modeling freedom” is covered primarily by the supply chain case study in chapter 6; “multiple fields of expertise” is covered primarily by the container terminal game in chapter 7; “language mismatch” is covered by the case studies of Nestlé production facilities in chapter 8.

Research question 2A was related to the difficulties to handle the unlimited modeling freedom by the model developer:

> *What constructs and design approach will enable that domain specific extensions can be defined independent of the generic simulation environment in such a way that the model developer is supported, but not limited to one way of representing a system element?*

In the case study of the supply chains (chapter 6) we extensively worked with different building block elements to represent parts of the flow in the supply chain, for example different ways of determining an order for a supplier, or the decision process to manufacture more products. The case study, using the implementations in different generic simulation environments, showed that the concept of simulation building blocks and building block elements applies and can support model developers. The answer to the question is thus:

> *Simulation building blocks and building block elements further supported by the guidelines of section 5.4 to ensure that the building blocks are self-contained, interoperable, reusable, replaceable, encapsulate their inner working, and provide useful services via precisely defined interfaces.*

Research question 2B was related to the challenge that model developers need to be experts in multiple areas without them knowing all about a domain. The model developer cannot know everything and cannot have talent and skills to cover all areas, therefore the second question was:

> *What methodologies, approaches and techniques can be offered to a model developer to support the use of domain specific extensions in the activities of a simulation study?*

In the case study of the container terminal it was shown that regular students without a deep knowledge of simulation or container terminals can be supported to perform a simulation study within 15 minutes and give an advice regarding the design of a container terminal, supported by quantitative and qualitative insights about the (logistical) performance of the designed container terminal. The key in achieving the 15 minute lead time was to automate as many actions in the activities of the simulation study as possible. The automation was achieved by a combination of structure in the input and data requirements, a structured simulation model and a structured representation of the output of the simulation model. The answer to research question 2B is therefore:

> *Additional tools to automate activities of model developers like model initiation and parameterization of the simulation model, and the translation of output data into key information.*

The final research question (2C) is related to the challenge that model developers do not speak the language of the problem owner. The model developer has different interpretations of the system, limited knowledge of the domain of the problem owner and is used to the generic terminology of the generic simulation environments, for example ‘resource’ or ‘queue’. In chapter 1 the question has been formulated as:

> *How can be ensured that the domain specific extension gets the model developer closer to the language of the problem owner?*

The set of simulation studies at different Nestlé production facilities showed the ability to represent the system using the terminology that the people in the factory are used to work with. The growth of the set of building blocks showed also that it is feasible to extend the scope of a domain specific extension and that this results in new terminology. The concept of simulation building block elements allowed to easily compose new dedicated simulation building blocks, therefore the problem owners in icecream factories noticed in their simulation model an icecream-filler while the problem owners of the petfood factory had in their model a petfoodbox-filler. Two machines that conceptually performed the same activities, but to the people involved in the simulation studies the specific attention to ‘their’ vocabulary was very

important. The answer to research question 2C as has been discussed more elaborate in chapter 5 and applied successfully in different case studies, but mainly in the Nestlé case study, is:

> *Use terminology of the problem owner in the simulation building blocks, example models and documentation.*

With the answers to the three sub questions and the successful execution of the case studies described in chapter 6, 7 and 8 and appendix I, the initial research question can be answered:

> *How can domain specific extensions for a simulation environment improve the effectiveness of simulation studies?*

The answer to this question is:

> *Perform simulation studies with models that are composed using domain specific extensions, by applying a structure* ***using simulation building blocks and building block elements****, include* ***additional tools*** *for automation, provide* ***support and documentation*** *for understanding via a* ***design approach*** *that focuses on conceptualization and specification.*

9.5 Further improvements and future research

Further improvements and future research regarding domain specific extensions can be seen in two directions. The first direction is to further improve the theory provided in chapter 5. At all four elements (concept of simulation building blocks, additional tools, support and documentation, and design approach) further improvements can be identified. Secondly, the availability of domain specific extensions enable a new way of carrying out simulation studies. The activities of a simulation study can be more focused, but also the new types of simulation studies can be defined. This section describes several of the potential research directions we have seen over the past years.

9.5.1 Improvements to the concept of simulation building blocks

The concept of simulation building blocks and the building block elements have been described in chapter 5. In the case studies we encountered that the simulation building block guidelines are applied differently according to the scope and level of detail of the simulation models within a domain. We saw that the guidelines have been applied in the Container Terminals case study. On the other hand, in the case studies for Supply Chains and Nestlé production facilities, the guidelines were applied more strictly. The experiences of the case studies show that the developers of the domain

specific extension have an interpretation of the guidelines and apply trade-offs whether a guideline should be applied or not.

Another point of attention is that the model developer makes trade-offs within the scope of the domain whether to use the guideline. These trade-offs deal with the decision whether the guideline should be applied to all building blocks, or only to a specific family. An example of such a decision is simulation building block guideline 2, see the block underneath with the trade-off explanation. Based on the experience of the performed case studies each simulation building block guideline can be extended in this way and further clarified to support the designers of domain specific extensions and their simulation building blocks.

Simulation Building Block guideline 2: a simulation building block consists of a core and building block elements to represent functions and services.

Trade-off explanation: In chapter 5 it has already been described that the simulation building block core can be coded with model constructs or the core can be represented by one or more building block elements. In the case studies described in chapter 6, 7 and 8, and in appendix 1, a rule of thumb has been applied to help in deciding between the core and the division over one or more building block elements. The rule of thumb consists of two parts. 1) functions that can be identified for potentially more than one building block should be modeled as a building block element. For example, claim equipment or release equipment. 2) if variants for these functionalities are important for the problem owner and if they result in different building blocks, then these functionalities should be implemented as separate building block elements. If not, they could be modeled in the core of the building block.

The case studies performed in chapter 6, 7 and 8 were executed with domain specific extensions that have been developed on top of three different generic simulation environments: Arena, eM-Plant and DSOL. With the case study of the supply chain (chapter 6) we demonstrated that one design for simulation building blocks and building block elements successfully can be implemented in different generic simulation environments. But the implementation project showed that some environments are easier to extend than others. We have combined our experience in working with the different simulation environments for the development of simulation building blocks and listed some features in Table 9.8. This table contains features that we observed in only one or two of the simulation environments. This table is intended for developers and vendors of generic simulation environments to apply these best practices into their simulation environment, resulting in the best generic simulation environment for the development of domain specific extensions.

Table 9.8: Features observed in one or two generic simulation environments

Feature	Generic sim. env.	Observed benefit that could also be realized in other simulation environments
(Partial) inheritance objects	eM-Plant	Ability to define a structure of objects, reusing parts of the functionality of superclasses, and possibilities for partial extension.
Composition of model constructs in model constructs	eM-Plant	Model constructs defined in the object library can be composed out of other model constructs. This enables the concept of building blocks composed out of building block elements.
Available data types	eM-Plant & DSOL	Availability of data types such as numbers, strings, tables, lists and possible new data types specific for the domain specific extension.
Visualization capabilities of state and performance indicators	Arena	The simulation building blocks that are instantiated into the simulation model automatically contain a state drawing, performance indicators and representation of parameter settings, which also scale when the model users zooms in or out.
Layered visualization	eM-Plant	The model developer has the opportunity to hide and show specific elements of the visualization of the building blocks, however, elements that are hidden at lower level, cannot be shown at a higher level of abstraction.
Capability of running via the web	DSOL	Simulation models developed in the DSOL environment can be accessed via any web browser and be executed and analyzed worldwide.
Programming interface with MS Office products	Arena	The additional tools developed for the container game are all programmed in Visual Basic and could be realized thanks to the complete integration of Visual Basic in the Arena simulation environment. This included model generation, parameter setting, model execution and reporting of performance indicators.
Capability of separate simulation models with one common clock	DSOL	The system elements are represented by several simulation models, divided over different computers, that all use the same central clock mechanism.
Routing of entities via stations	Arena	Entities with all their attributes are sent from station to station, where at each station, the correct code can be executed, based on the state of the attributes of the entity that resides in the mailbox.

Feature	Generic sim. env.	Observed benefit that could also be realized in other simulation environments
Connect model constructs in flow	Arena	The Nestlé process building blocks were fluently connected via connector interfaces that enable the representation of a flow analogous to how process engineers use their flow schemes of the production mechanism.
Debugging capability	eM-Plant & DSOL	While the simulation model is in debug mode, the code can be viewed exactly as it is programmed, it can be executed step wise, and it is possible to set breakpoints and return to breakpoints.
Building block changes during simulation run mode	eM-Plant	While the simulation model is running, it is possible for expert model developers to temporarily stop the clock, adjust the simulation model logic and continue. This change can also be applied to the simulation building blocks that are instantiated in the simulation model. Especially during the system testing of the simulation building blocks for initial development this feature is very helpful for developers.

9.5.2 Improvement to additional tools

The additional tools defined in chapter 5 are simulation model instantiation, model parameterization, model verification and analysis of the model output. In the container game all these types of tools have been developed from scratch. The domain specific extension for Nestlé production facilities included with the Excel sheet only the parameterization and the output analysis. All of these tools have been developed from scratch as well, even though the simulation building blocks of the domain specific extension were both implemented using the generic simulation environment Arena.

The parameter interface of simulation building blocks developed in Arena all have the same structure, therefore it should be feasible to have a standardized approach of filling the parameters. The same applies for the output of the simulation model that is defined in text files. A standard reusable approach should be available to handle the output of simulation building blocks.

The standardized approach should contain a way of handling the output of simulation building blocks and the best way to represent it. Some experience was gained within the Nestlé simulation studies with combining and abstracting performance indicators. On the other hand, the lack of combining and abstracting was one of the major concerns in the teaching cases in the supply chain (Van der Hee, 2002). The definition of additional tools could lead to a better standardization and conceptually match better with the use of the data and analysis on behalf of the problem owners.

9.5.3 Improvement to support and documentation

In all of the three key domain specific extensions of chapter 6, 7 and 8, a lot of effort was put to the documentation of the individual simulation building blocks. This documentation contained the functionality of the individual simulation building blocks, a detailed description of their parameters, the visualization, the performance indicators and additional background information.

In the training sessions provided to model developers and users of the different domain specific extensions, the model developers received sufficient explanations with the functionality descriptions of the simulation building blocks. With that knowledge they successfully applied the building blocks in training assignments. They hardly ever paid attention to the other descriptions of the simulation building blocks and just started working. Further research is needed to develop a more targeted documentation, support, and training sessions, and thus a more efficient investment in the domain specific extension.

9.5.4 Improvement to the design approach for domain specific extensions

If the simulation building block guidelines are extended with trade-offs, then these trade-offs will enhance the design approach for domain specific extensions, especially when the trade-offs contain enhanced mitigations for the list of risks that have been identified. The trade-offs and the use of the simulation building blocks guidelines can then become an integrated part of the design approach. The advantage of an integrated design approach is that the designer has a step-wise approach, including a descriptions of workshops and intermediate documents, which help the complete design team and problem owner to understand the process, the need of contribution and the risks involved in insufficient participation by making undesired trade-offs.

9.5.5 New possibilities simulation and challenges using domain specific extensions

The domain specific extensions that have been developed in the case studies enabled fast and easy model development, partially automatic setting of the parameters and in two cases complete automatic model instantiation, i.e. container game from VISIO drawings (chapter 7) and airport baggage systems from AutoCAD drawings (appendix 1). These structured simulation models allow for some new possibilities, beyond the used of default problem solving as described in section 1.2.

The first experiences of using simulation models developed with domain specific extensions for other purposes than the intended simulation study for problem solving and alternative experimentation were observed in the case studies of OLS (control of hardware from simulation, section 3.3), supply chain (use of real time data, section 6.6), container design (applicability in a

management game, chapter 7), Nestlé production facilities (operational planning optimization and generation of PLC logic, section 8.6) and baggage handling at airports (test control software via simulation, appendix 1) possibly automatic model development.

The potential of these topics can be enormous in reducing the cycle of investments and the quality of the operational performance of the investment. However, each of these topics for further use of simulation models has its own challenges. The potential and the main challenges that have been observed during the execution of the case studies are briefly discussed as ongoing follow up research.

Possibility of using real time data

The potential of real time data in simulation models is that an actual situation can be reviewed in the model and to have an accurate starting point to predict what the system will look like when extrapolated from its current situation. In the supply chain case study this resulted in a model that included simulated events as well as actual events. Jacobs (2005) describes some concepts regarding the generic simulation environment how to handle this. Likely also some (additional) simulation building block guidelines are required to incorporate this in the domain specific extension.

Possibility of using PLC linking

Saanen (2004) describes how the process of readying an automated container terminal for production is performed, and the enormous challenges the developers have with the testing the PLC code. He describes how simulation models could be used to test the PLC code by interfacing between the simulation software and the PLC software. The OLS project (chapter 3) and the baggage handling at airport (appendix) are case studies that helped Saanen in the integration and testing.

Possibility of supporting games with simulation

In the management game ‘Container Adrift’ the visualization-simulation tool is used to support the negotiations around a terminal design. This enabled the stakeholders to have discussion about the content besides discussion about the processes. Applying simulation models in this way increases the quality of the discussion. The challenge for the game participants is not to lose themselves in the details offered by the tool. Van Houten (2007) has used this idea and the supply chain simulation building blocks to develop a design approach for games. Some specific building block elements could be defined to enhance the process of game development.

References

Ackoff, R.L. (1962) *Scientific method: optimizing applied research decisions*. New York: John Wiley & Sons.

Arends, D. (1999) *Using object-oriented simulation for a quantitative approach of the terminal concepts,* master thesis, Delft University of Technology, Delft

Ayad N.; H.G.Sol. (2002) “Development of New Geographically Distributed Business Models for Global Transactions”. In: R.H. Sprague, J.F. Nunamaker (Eds.); *Proceedings of the 35th Hawaiian International Conference on Systems Sciences*, CD-ROM, 8 pages

Ayad, N.; E.C. Valentin. (2001) *Design building blocks for global banking at ABN AMRO,* intern report, Delft University of Technology, Delft

Babeliowsky, M.N.F. (1997) *Designing interorganizational logistic networks, a simulation based interdisciplinary approach,* Doctoral Dissertation, Delft University of Technology, Delft

Bajnath, S.S.; S.J. Bani Hashemi (2007) *DSOL For Hospitals; A simulation study on using DSOL in a Web-based Environment for Hospitals*, intern report, Delft University of Technology, Delft

Baker, G. (1997) “Taking the work out of simulation modeling: an application of technology integration” In: S. Andradottir; K.J. Healy; D.H. Withers; B.L. Nelson (Eds.) *Proceedings of the 1997 Winter Simulation Conference*, pp.1345-1351

Balci, O. (1997) “Principles of simulation model validation, verification, and testing” In: *Transactions of the Society for Computer Simulation International*, 14 (1), pp.3-12

Balci, O; R.E. Nance. (1992) “The simulation model development environment: an overview” In: J.J. Swain; D. Goldsman; R.C. Crain; J.R. Wilson (Eds.) *Proceedings of the 1992 Winter Simulation Conference*, pp.726-737

Banks, J. (1999) “Introduction to simulation”, In: P.A. Farrington; H.B. Nembhard; D.T. Sturrock ; G.W. Evans (Eds.) *Proceedings of the 1999 Winter Simulation Conference*, pp.7-13

Banks, J. (2000) *Getting started with Automod*. Salt Lake City: Brooks Automation

Banks, J.; F. Azadivar; D. Ferring; J.W. Fowler; D.W. Halpin; A.M. Law; M. Manivannan; W.S. Murphy. (2001) “Panel session: the future of simulation” In: B.A. Peters; J.S. Smith; D.J. Medeiros; M.W. Rohrer (Eds.) *Proceedings of the 2001 Winter Simulation Conference*, pp.1453-1461

Bapat, V.; D.T. Sturrock. (2003) “The Arena product family: enterprise modeling solutions” In: S. Chick; P.J. San-chez; D. Ferrin; D.J. Morrice (Eds.) Proceedings of the 2003 Winter Simulation Conference, pp.210-217

Barton, R.R.; P.A. Fishwick; R.G. Sargent; J.O. Henriksen; J.M. Twomey. (2003) “Panel: simulation – past, present and future” In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.2044-2050

Batley, A. (2006) “Simulation experiments for roadmap of investment in milk factory” In: R.A.Bijlsma (Ed.); *Proceedings of ArenaSphere 2006*, CD-ROM, 12 pages

Birtwistle, G.M. (1979) *DEMOS, a system for discrete event modeling on SIMULA.* London: Macmillan

Blom, P.; J. Korf. (2000) *Bottleneck analysis of the new terminal of JFKIAT*. Delft: TU Delft

Bockstael-Blok,W.; I.S. Mayer; E.C. Valentin. (2003) “Supporting the design of an inland container terminal through visualization and gaming-simulation” In: R.H. Sprague, J.F.Nunamaker (Eds). *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, CD-ROM , 10 pages

Boyson, S.; T. Corsi; M. Dresner; L. Harrington. (1999) *Logistics and the Extended Enterprise*, New York: John Wiley & Sons Inc.

Brandt, M. (1999) *SkyShuttle, Den Haag in beweging* (Dutch), master thesis, Delft University of Technology, Delft

Britals, J. (2008) “Enterprise Dynamics New Release V8.0” In: S.J. Mason; R.R. Hill; L.Mönch; O Rose; T.Jefferson; J.W.Fowler (Eds.) *Proceedings of the 2008 Winter Simulation Conference*, pp.215-221

Bruijn, H. de; E. ten Heuvelhof. (2000) *Networks and Decision-making*. Utrecht: Lemma

Cellier, F.E. (1992), “Hierarchical Non-Linear Bond Graphs: A Unified Methodology for Modeling Complex Physical Systems”, *Simulation*, 58(4), pp.230-248.

Cooper, M.; D. Lambert; J. Pagh. (1997) “Supply chain management: More than just a name for logistics” In: *The International Journal of Logistics Management*, 8 (1), pp.1-14

Corver, A. (2001) *Supply chain visualization: Simulation as a means to gain insight in the supply chain*. master thesis, Delft University of Technology, Delft

CTT-Center for Transportation Technology. (1997) *Definitiestudie ondergronds logistiek systeem, deelrapportages 1-5.* Rotterdam: CTT

Dahl, O.-J.; K. Nygaard (1966) “SIMULA - an ALGOL-Based Simulation Language”. *Communications of the ACM*, 9(9), pp. 671-678

Davis, P.C.; P.A. Fishwick; C.M. Overstreet; C.D. Pegden. (2000) “Model composability as a research investment” In: J.A. Joines; R.R. Barton; K. Kang; P.A. Fishwick (Eds.) *Proceedings of the 2000 Winter Simulation Conference*, pp.1585-1591

Diamond, R.; J.O. Henriksen; C.D. Pegden; A.P. Walker; C.R. Harrell; W.B Nordgren; M.W. Rohrer; A.M. Law. (2002) “The current and future status of simulation software (panel)” In: E. Yücesan; C.H. Chen; J.L. Snowdon; J.M. Charnes (Eds.) *Proceedings of the 2002 Winter Simulation Conference*, pp.1633-1640

Du, W. (2002a) *Ontwikkeling bouwstenen voor simulatie-spel*. master thesis, Delft University of Technology, Delft

Du, W. (2002b) *User manual visualization-simulation tool Containers Adrift*. Delft University of Technology, Delft

Duke, R. (1980) “A paradigm for game design”, *Simulation and Gaming*, 11(3), pp.364-377

Ebben, M.J.R. (2001) *Logistics Control in Automated Transportation Networks.* doctoral dissertation, University of Twente, Enschede

Eldabi, T.; M. Wai Lee; R.J. Paul (2003) “A Framework For Business Process Simulation: The Grab And Glue Approach”. In: A. Verbraeck, V. Hlupic (Eds.), *Proceedings ESS’2003 – 15th European Simulation Symposium,* pp.141-148

Evers, J.J.M.; S.A.J. Koppers. (1996) “Automated guided vehicle traffic control at a container terminal”, *Transportation research part A – Policy and practice*, 30 (1), pp. 21-34

Flood, R.L.; E.R. Carson. (1988) *Dealing with Complexity*. New York: Plenum Press

Forrester, J.W. (1999) *System Dynamics: the Foundation Under Systems Thinking.* Cambridge: MIT

Gast, V. de ; R.A. Bijlsma; E.C. Valentin. (2008) “Empowering Decision Support With Simulation Technology – Scenario Navigator” In: S.J. Mason; R.R. Hill; L.Mönch; O Rose; T.Jefferson; J.W.Fowler (Eds.) *Proceedings of the 2008 Winter Simulation Conference*, pp.236-244

Gatersleben, M.R.; S.W. van der Weij. (1999)“Analysis and simulation of passenger flows in an airport terminal”. In: P.A.Farrington, H.B.Nembhard, D.T.Sturrock and G.W.Evans (Eds.), *Proceedings of the 1999 Winter Simulation Conference*, pp.1226-1231

Gigch, J.P. van (1991) *System design modeling and metamodeling*. New York: Plenum Press

Haige, J.W.; K.N. Paige. (2004) *Learning SIMUL8: The Complete Guide, 2nd Ed*. Bellingham: Plain Vu Publishers

Harrell, C.R.; R.N. Price. (2003) “Simulation modelling using Promodel Technology” In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) Proceedings of the 2003 Winter Simulation Conference, pp.175-181

Hay, A.M.; E.C. Valentin; R.A. Bijlsma. (2006) “Modeling Emergency Care in Hospitals: A Paradox - The Patient Should not Drive the Process.” In: E.Yücesan; C.H.Chen; J.L.Snowdon; J.M.Charnes (Eds.) *Proceedings of the 2006 Winter Simulation Conference*, pp.439-445

Hee, R. van der. (2001) *Teaching cases Real-Time Supply Chain*. CD-ROM, Delft University of Technology

Hee, R. van der. (2002) *Building blocks for Real-Time Supply Chain,* master thesis, Delft University of Technology, Delft

Heijden, M.C. van der; A. van Harten; M.J.R. Ebben; Y.A. Saanen; E.C. Valentin; A. Verbraeck. (2002) “Using Simulation To Design an Automated Underground System for Transporting Freight Around Schiphol Airport”, *Interfaces*, 32 (4), pp.1-19

Heijman, F. (1999) *Check-in capacity analysis of Terminal 4 at JFK*, master thesis, Delft University of Technology, Delft

Heinicke, M.U.; A. Hickman. (2000) “Eliminate bottlenecks with integrated analysis tools in eM-Plant” In: J.A. Joines, R.R. Barton, K. Kang, P.A. Fishwick (Eds.) *Proceedings of the 2000 Winter Simulation Conference*, pp.229-231

Hill, D.R.C. (1996) *Object-Oriented Analysis and Simulation* Bedforshire: Addison-Wesley

Hooghiemstra, J.S.; M.J.G. Teunisse. (1998) “The use of simulation in the planning of the Dutch railway services” In: D.J. Medeiros; E.F. Watson; J.S. Carson; M.S. Manivannan (Eds.) *Proceedings of the 1998 Winter Simulation Conference*, pp.1139-1145

Hospital Navigator. (2007) *User’s Guide: Hospital Navigator; Modelling, Simulation & Analysis in Healthcare,* Den Haag: Systems Navigator

Houten, S.P.A. van (2007) *A suite for developing and using business games*, doctoral dissertation, Delft University of Technology, Delft

Jacobs, P.H.M. (2005) *DSOL simulation suite - Enabling multi-formalism simulation in a distributed context.* doctoral dissertation, Delft University of Technology, Delft

Kalasky, D.R., Levasseur, G.A. (1997). “Using SiMPLE++ for Improves Modeling Efficiencies and Extending Model Life Cycles” In: S. Andradottir; K.J. Healy; D.H. Withers; B.L. Nelson (Eds.) *Proceedings of the 1997 Winter Simulation Conference*, pp.1345-1351

Kasputis S.; H.C. Ng. (2000) “Composable simulations” In: J.A. Joines; R.R. Barton; K. Kang; P.A. Fishwick (Eds.) *Proceedings of the 2000 Winter Simulation Conference*, pp.1577-1584

Kempen, J. van ; I.S. Mayer; W. Bockstael-Blok. (2002) *Spel beschrijving Containers op Drift*, Delft University of Technology, Delft

Keller, L.; C. Harrell; J. Leavy.(1991) “The three reasons why simulation fails” *Industrial Engineering,* 23(4), pp.27-31

Kelton, W.D.; R.P. Sadowski ; D.T. Sturrock. (2003) *Simulation with Arena, third edition.* New York: McGraw-Hill

Kim, T.G.; Ang, M.S., (1997) “Chapter 4. Reusable Simulation Models in an Object-Oriented Framework”. In: Zobrist, G.W.; Leonard, J.V. (Eds.) *Object-oriented simulation: reusability, adaptability, maintainability.* pp. 139-164.

Kiviat, P. J., (1966) *Introduction to the SIMSCRIPT II Programming Language*. New York: The RAND Corporation

Kolfschoten, G.L. ; E.C. Valentin; G.J. de Vreede; A. Verbraeck. (2006) “Cognitive load reduction through the use of building blocks in the design of decision support systems”, In: *Proceedings of the Twelfth Americas Conference on Information Systems*, CD-ROM.

Kolfschoten, G.L.; S. Lukosch; A. Verbraeck; E.C. Valentin; G.J. de Vreede (2010) “Cognitive learning efficiency through the use of design patterns in teaching.” *Computers & Education*, 54(3), pp. 652-660

Krahl; D. (2003) “Extend: an interactive simulation tool” In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference,* pp.188-196

Kuiper, R. (2001) *Fly or Flee, Evacuation of Airport Terminal Buildings – Possibilities for Computer Simulation*, master thesis, Delft University of Technology, Delft

Law, A.M.; D.W. Kelton. (1999) *Simulation Modeling and Analysis.*New York : McGraw-Hill

Law, A.M.; M.G. McComas. (1989) “Pitfalls to Avoid in the Simulation of Manufacturing Systems” *Industrial Engineering*, 21(5), pp.28-31

Lindeijer, D.G. (2003) *Controlling Automated Traffic Agents*, doctoral dissertation, Delft University of Technology, Delft

Mayer, I.S., W. Bockstael-Blok, E.C. Valentin. (2004) “A Building Block Approach to Simulation: An Evaluation Using Containers Adrift” In: Simulation Gaming, 35 (1), pp 29-52

McClave, J.T.; P.G.Benson; T.L. Sincich. (2000) *Statistics for Business and Economics (8th Edition)*, Upper Saddle River: Prentice Hall PTR

Meer, T. van der. (2002) “Logistieke megaprojecten op lange baan geschoven” *LogistiekKrant*, 15 (17), pp. 1

Mehta, A. (1999) “Business solutions using Witness” In: P.A. Farrington; H.B. Nembhard; D.T. Sturrock; G.W. Evans (Eds.) *Proceedings of the 1999 Winter Simulation Conference*. pp.230-233

Mitroff, I.I.; F.R. Sagasti. (1973) “Operation research form the viewpoint of general systems theory” *OMEGA*, 1 (6), pp.117-134

Nance, R.E. (1993) “A history of discrete event simulation programming languages” *ACM SIGPLAN Notices*, 28 (3), pp 149-175.

Page, E.H.; J.M. Opper. (1999) “Observations on the complexity of composable simulation” In: P.A. Farrington; H.B. Nembhard; D.T. Sturrock; G.W. Evans (Eds.) *Proceedings of the 1999 Winter Simulation Conference*, pp.553-560

Pater, A.J.G.; M.J.G. Teunisse.(1997) “The use of a template-based methodology in the simulation of a new cargo track from Rotterdam harbor to Germany” In: S. Andradottir; K.J. Healy; D.H. Withers; B.L. Nelson (Eds.) *Proceedings of the 1997 Winter Simulation Conference*, pp.1176-1180

Pegden, C.D.; R.E. Shannon; R.P. Sadowski. (1990) *Introduction to Simulation Using SIMAN.* New Jersey: McGraw-Hill

Pielage, P. (2005) *Conceptual Design of Automated Freight Transport Systems. Methodology and Practice*, doctoral dissertation, Delft University of Technology, Delft

Pollacia, L.F.; L.M.L. Delcambre. (1997) “Chapter 3. The object flow model for object-oriented simulation and database application modeling”. In: G.W. Zobrist; J.V. Leonard (Eds.) *Object-oriented simulation: reusability, adaptability, maintainability.* Piscataway: IEEE Press, pp. 89-137.

Rengelink, W.; Y.A. Saanen. (2002) “Improving the Quality of Controls and Reducint Costs for On-Site Adjustments with Emulation: An Example of Emulation in Baggage Handling”. In: E.Yücesan; C.H.Chen; J.L.Snowdon; J.M.Charnes (Eds.) *Proceedings of the 2002 Winter Simulation Conference*, pp.1689-1694

Riesenkamp, M.; E.C. Valentin; G. Heijkoop; C. Konings. (2007) *Functionele Omschrijving: Sandd – Simulatie en experimentatie – capaciteitsplanning.* Intern rapport, Systems Navigator, Delft

Robinson, S.; M. Pidd. (1998) “Provider and customer expectations of successful simulation projects” *Journal of the Operational Research Society*, 49(3), pp.200-209

Rockwell Automation (2007) “Arena Shines at WSC—Again!” Retrieved 20-March-2007 from: www.arenasimulation.com

Rohrer, M.W. (2003) “Maximizing simulation ROI with AutoMod” In: S. Chick; P.J. Sanchez; D. Ferrin; D.J. Morrice (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, pp.201-209

Saanen, Y.A. (2004) *An approach for designing robotized marine container terminals*, doctoral dissertation, Delft University of Technology, Delft

Sadowski, D.; M.R. Grabau. (2000) “Tips for successful practice of simulation” In: J.A. Joines; R.R. Barton; K. Kang; P.A. Fishwick (Eds.) *Proceedings of the 2000 Winter Simulation Conference*, pp.69-76

Sage, A.P. ; J.E. Armstrong. (2000) *Introduction to systems engineering*. New York: John Wiley & Sons

Schriber, T.J. (1974) *Simulation using GPSS.* New York: John Wiley

Shannon, R.E. (1975) *Systems simulation: the art and science.* Upper Saddle River: Prentice Hall PTR

Simon, H.A. (1969) *The sciences of artificial*. Cambridge: MIT Press

Sol, H.G. (1982) *Simulation in information systems development*, Doctoral Dissertation, Rijksuniversiteit Groningen, Groningen

Spruengli, G.; E.C. Valentin; S. Steijaert (2005) *Report workshop scoping simulation building blocks Nestlé Nutrition*. Geneva: Rockwell Automation

Swain, J.J. (2007). "INFORMS simulation software survey. OR/MS Today. Institute for Operations Research and the Management Sciences (INFORMS), USA." Retrieved 02-March-2008 from: http://www.lionhrtpub.com/orms/surveys/Simulation/Simulation.html

Systems Navigator (2005) *User Manual Model Seine Nord*, Den Haag: Systems Navigator

Systems Navigator (2006) *User's Guide: Business Process Simulation Building Blocks; Modeling Simulation & Analysis of business process*. Den Haag: Systems Navigator

Systems Navigator (2007) *User Manual Nestlé Simulation Building Blocks, version 2*, Delft: Systems Navigator

Tewoldeberhan, T.W., (2005) *Gaining insight into business networks; A simulation based support environment to improve process orchestration*, doctoral dissertation, Delft University of Technology, Delft

Turner, M.J. (2006) "Designing UK Government Shared Services" In: *Proceedings European ArenaSphere 2006*, CD-ROM

Valentin, E.C. (2002) "Building blocks for modeling of passengers at airports" In: A. Verbraeck; A. Dahanayake (Eds.) *Building blocks for Effective Telematics Application Development and Evaluation* pp.148-172

Valentin, E.C., I.S. Mayer, W. Bockstael-Blok. (2002) "Simulate designs of container terminals in 15 minutes". In: A. Verbraeck, W. Krug (Eds), *Simulation in Industry – 14th European Simulation Symposium 2002*. pp 308-312

Valentin, E.C.; R. Sadowski (2003) "Criteria system analysis of a simulation study in transportation domain", internal report, Delft University of Technology

Valentin, E.C.; A. Verbraeck; H.G. Sol. (2003a) "Effect of Simulation Building Blocks On Simulation Model Development" In: A. Ibarra (Ed.) *Proceedings of International Conference of Technology, Policy and Innovation*, pp.54-61

Valentin, E.C.; A. Verbraeck; H.G. Sol. (2003b) “Advantaged and disadvantages of building blocks in simulation studies: a laboratory experiment with simulation experts” In: A. Verbraeck, V. Hlupic (Eds.), *Proceedings ESS’2003 – 15th European Simulation Symposium,* pp.141-148

Valentin, E.C.; S. Steijaert; R.A. Bijlsma; P. Silva. (2005a) “Approach for Modelling of Large Maritime Infrastructure Systems”, In: M.E. Kuhl; N.M. Steiger; F.B. Armstrong; J.A. Joines (Eds.) *Proceedings 2005 Winter Simulation Conference*, pp.1577-1585

Valentin, E.C.; S. Steijaert; G. Spruengli. (2005b) *Brainstorming for Nestlé specific simulation*. Den Haag: Systems Navigator

Valentin, E.C.; D. Gstoehl; A. Batley; A. Richoz; R.A. Bijlsma. (2005c) *Specification Nestlé Specific simulation building blocks*. Den Haag: Systems Navigator

Valentin, E.C. (2007) *Conceptual model description Ice cream factory Rorschach* Den Haag: Systems Navigator

Valentin, E.C.; A. Verbraeck. (2007) “Domain specific model constructs in commercial simulation environments”. In: S.G. Henderson; B. Biller; M.-H. Hsieh; J. Shortle; J.D. Tew; R.R. Barton (Eds.) *Proceedings of the 2007 Winter Simulation Conference.* pp. 785-795

Valentin, E.C.; A. Nati; T. Holt; G. Piot; R.A. Bijlsma. (2007) *Extension specification Nestlé Specific simulation building blocks advanced processes*. Den Haag: Systems Navigator

Vangheluwe, H.L.; G.C. Vansteenkiste. (1997) “Multi-formalism modelling and programming language types”. In: W. Hahn ; A. Lehmann (Eds.), *Simulation in Industry*, Society for Computer Simulation International (SCS), pp.105-109

Vangheluwe, H.L.; J. de Lara. (2002) “Meta-models are models too” In: E. Yücesan; C.H. Chen; J.L. Snowdon; J.M. Charnes (Eds.) *Proceedings of the 2002 Winter Simulation Conference*, pp.597-605

Verbraeck, A.; Y.A. Saanen; E.C. Valentin. (1998a) “Logistic Modeling and Simulation of Automated Guided Vehicles”. In: A.Bargiela, E.Kerckhoffs (Eds.) *Simulation Technology: Science and Art. 10th European Simulation Symposium and Exhibition.* pp.514-519

Verbraeck, A.; Y.A. Saanen; E.C. Valentin. (1998b) *Terminal rapportage Simulatie Experimenten-deel 1*(Dutch). Delft: Connekt

Verbraeck A.; Y.A. Saanen; E.C. Valentin. (1999) "Simulatie voor het Ondergronds Logistiek Systeem Schiphol (Dutch)". In: J.van Nunen, Leonard Verspui (Eds.*). SimLog: Simulatie en Logistiek rond de Haven*, pp.145-156

Verbraeck, A. (2002) “Chapter 1: The BETADE Research Program”. In: A. Verbraeck; A. Dahanayake (eds.) *Building blocks for Effective Telematics Application Development and Evaluation,* pp.2-7

Verbraeck A.; Y. Saanen; Z. Stojanovic; B. Shishkov; A. Meijer; E. Valentin; K. van der Meer. (2002) “Chapter 2: What are building blocks?”. In: A. Verbraeck; A. Dahanayake (eds.) *Building blocks for Effective Telematics Application Development and Evaluation,* pp.8-21

Verbraeck, A. (2004) “Real-time simulation for real-time supply chains.” In S. Boyson; L.H. Harrington; T.M. Corsi (Eds.), *Real Time: Managing the new Supply Chain*; pp. 99-122

Versteegt, C. (2004) *Holonic control for large scale automated logistic systems*, doctoral dissertation, Delft University of Technology, Delft

Visser, R.J. (2000) *Sturen zonder handen* (Dutch), master thesis, Delft University of Technology, Delft

Vreede, G.J. de; E.C. Valentin; J.C. Schuuring. (2002) *Eindrapportage "Resultaat door Kleinschaligheid".* Rotterdam: DITSE

Witt-Hamer, K. de (1999) *Managing the check-in process*, master thesis, Groningen University, Netherlands

XJ Technologies. (2005) *ANYLOGIC*^{tm} St Petersburg: User’s Manual, St Petersburg, 2005

Zeigler, B.P.; H. Prähofer; T.G. Kim. (2000) *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. San Diego: Academic Press

Zobrist, G.V.; J.V. Leonard. (1997) *Object-Oriented Simulation – Reusability, Adaptability, Maintainability.* New York: Wiley-IEEE Press

Appendix 1: Additional cases with domain specific extensions

Domain specific extensions have been developed for several more domains in the period from 2002 to 2007 than only the domain for supply chains (chapter 6), container terminals (chapter 7) and Nestlé production facilities (chapter 8). The domain specific extensions were designed according to the design approach described in section 5.7, the guidelines for simulation building blocks described in section 5.4 and the use of additional tools and support to model developers as mentioned in section 5.5 and 5.6.

The author of this thesis has participated in the development and use of these domain specific extensions, and therefore he assured the use of the guidelines and approaches as mentioned. However, the domain specific extensions have not been used as extensive as the three examples described in the main text. The use has been limited to only one simulation study or the model development has only been performed by the developer of the domain specific extension. Nevertheless, the case studies describe the wide applicability of the concept, guidelines and design approach and therefore they have been included in this appendix.

Appendix 1.1 Passengers at airport, once more

In section 3.3 we described three simulation studies of passengers at airports. In those simulation studies the number of model constructs rapidly extended due to small differences in functionality of model constructs. In these simulations there also were difficulties with the modeling of the allocation algorithms to allocate scarce resources such as gates or check-in counters to flights.

We redeveloped the domain specific extension in the simulation environment eM-Plant with the theory described in chapter 5 and the knowledge encountered in the three simulation studies of passengers at airports described in section 3.3. In the new domain specific extension, more attention was put to functionalities of system elements that were differing or shared between the system elements.

The infrastructure represented by an ‘Area’ has been modeled by several building block elements representing tasks like ‘Allow passenger to enter area’, ‘Manage capacity in area’ or ‘Determine process duration in area’. These types of building block elements went through several implementations resulting in more flexibility in representing an area, and easier changes in the simulation models (Valentin, 2002).

The use of building block elements also improved the flexibility of using specific algorithms. The first simulation study performed with the new domain specific extension for passengers in airports regarded the planning and

managing of personnel at the security and passport checks at Amsterdam Airport Schiphol (Visser, 2000). The model developers required a specific algorithm of allocating personnel to the areas for passport checks or security checks. Visser solved this by extending the domain specific extension with new building block elements to change the capacity of an area based on an external trigger and a building block element to dynamically reroute passengers that were queuing.

The second simulation study with the improved domain specific extension for passengers at airports also included the development of new building block elements. Kuiper (2001) describes a research project together with the Airport Research Center (www.arc-aachen.de) for the evacuation of passengers from an airport in case of emergencies. He extended the areas with a building block element that represents an alarm or intercom message that informs all passengers to leave the airport as soon as possible. The new building block elements in the domain specific extension enabled simulations to represent different alarming mechanisms and to study the effects of alternative emergency exits in an airport building. Kuiper used the new building block elements (Figure Appendix.1) in simulation models of different airports to prove their applicability.

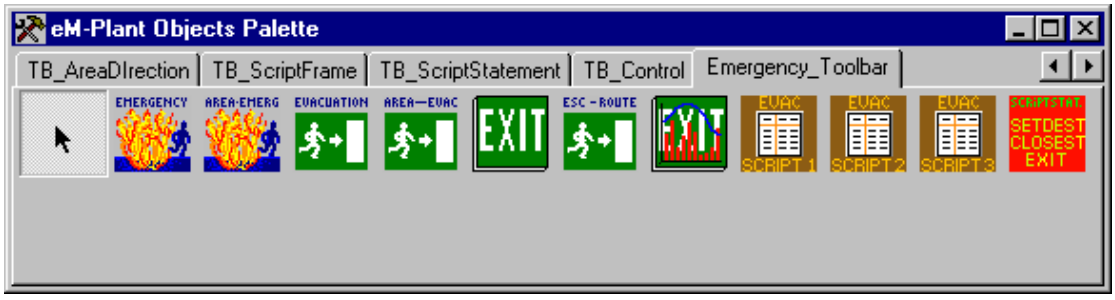


Figure Appendix.1: New building block elements added to the domain specific extension for passengers in airports (Kuiper, 2001, p78)

Appendix 1.2 Baggage handling at airports

Baggage systems become an essential element in the quality that an airport provides to its customers. Just a single conveyor belt and manually sorting of baggage is not possible, due to operational costs and available time between arrival of passenger and departure of a plane. Airports require from vendors to have a simulation study that proofs that their design will be able to handle the daily peaks.

Many vendors have invested in new mechanisms of transporting baggage, other than using conveyor belts. Systems like automatic vehicles or magnetic steered carts require high investments and thus a trade-off between flexibility and investment. The investment in overcapacity can be reduced by smart control mechanisms and just in time arrival of vehicles for transporting

baggage. This can only be achieved with advanced management of the operations in the baggage area.

A team of simulation experts and baggage experts worked together at the development of a domain specific extension for modeling of baggage systems. The input from the experts of the TU Delft was generic simulation knowledge, experts of TBA provided mainly knowledge of simulating baggage systems and experts of the Airport Research Center provided knowledge of animating and drawing baggage systems using AutoCAD.

The challenge in this domain specific extension was to develop a set of building blocks in the simulation environment eM-Plant including all infrastructure and control of the infrastructure as well as a set of building blocks in AutoCAD to easily translate a drawing to a simulation model for analysis. The simulation model after the analysis should then be translated back to the AutoCAD drawing for 3D animation and finalizing engineering activities.

Another expected use of the domain specific extension is the use of different layers for infrastructure, control and management, which allows interfacing with real-time objects. This should be a follow up of the successful exchange between simulation model and reality as demonstrated in the simulation project for the OLS (section 3.2).

This domain specific extension consists of different types of transportation, among them conveyors, automatic vehicles and magnetic transponders. The simulation building blocks are composed from a range of building block elements to enable easy development of new pieces of infrastructure, e.g. a new type of vehicle or a new crossing between conveyors.

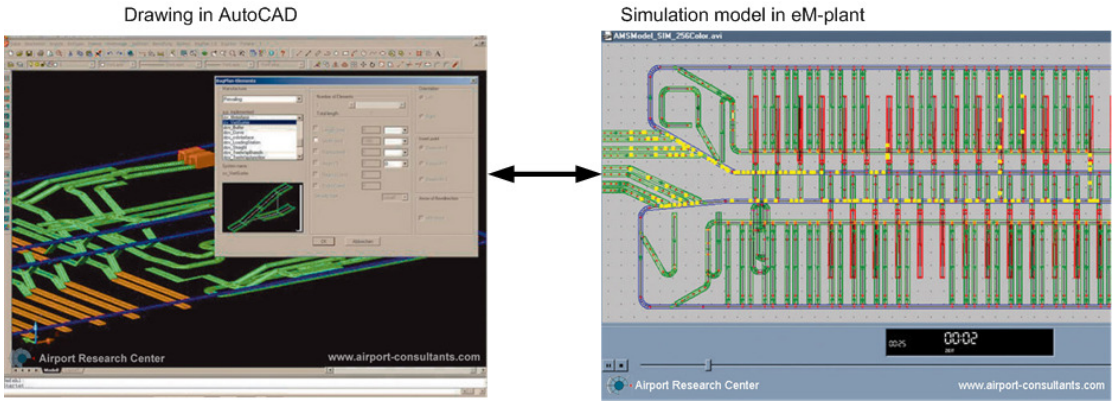


Figure Appendix.2: Representation of baggage handling system as drawing and simulation model (www.arc-aachen.de)

The interaction between AutoCAD and eM-Plant worked out as expected thanks to the same definition of building blocks in the AutoCAD environment as in the eM-Plant object libraries. As a result a simulation model can be converted to an AutoCAD drawing and an AutoCAD drawing can be converted to a simulation model. This interaction is shown in Figure Appendix.2. At the left hand side the AutoCAD drawing is shown of baggage handling system in Greece and at the right hand side the instantiated simulation model of this

system. Changes that are made in the AutoCAD drawing via the dedicated user interface (shown also in Figure Appendix.2) will automatically result in changes to the simulation model.

The interaction between simulation models and the control software of physical baggage equipment has not been realized in a real life case, but has been performed in smaller laboratory settings in which several conveyors belts were controlled via a simulation model. Rengelink and Saanen (2002) describe how they test PLC logic via the simulation model by replacing the control of the simulation by external PLC logic.

Besides the project for testing of PLC logic (Rengelink and Saanen, 2002) the domain specific extension for baggage handling has been applied at several airports. Among them are the airport of Athens (Greece) and the Moscow-Sheremetyevo Airport (Russia) (www.arc-aachen.de).

Appendix 1.3 Business processes for shared service centers

The UK government is in progress of introducing shared service centers for back office activities like human resources, finance and procurement. The aim of introducing the shared service centers is to standardize procedures, reduce the number of employees involved by 67% and improve throughput. These aims will be achieved by introducing administrative tools and by restructuring departments and task allocations. Discrete event simulation is used to support in defining the required number of employees, the size of work teams and provide qualitative information of lead times and pieces of work in progress (Turner, 2006).

The design of the shared service centers is performed in 8 cycles, representing different tasks within the back offices. The first step is a new design of the processes in the shared service center. The second step is to develop a simulation model of this process and finally to perform simulation experiments to determine the size of work teams and required skills of employees by varying task allocations.

Simulation models of cycle 1 and 2 of the design of the shared service have been developed with model constructs of the generic simulation environment. These simulation models represented the system as designed, but had the disadvantage that, due to the available model constructs, there were quite some differences between the VISIO design and the process flow in the simulation model.

In the domain specific extension for the container terminals we designed building blocks in VISIO and in the simulation environment (chapter 7). In this domain specific extension the exchange should not be for equipment, but process related. The challenge is that the process in a VISIO chart represents different levels and that these different levels result in combined and grouped statistics.

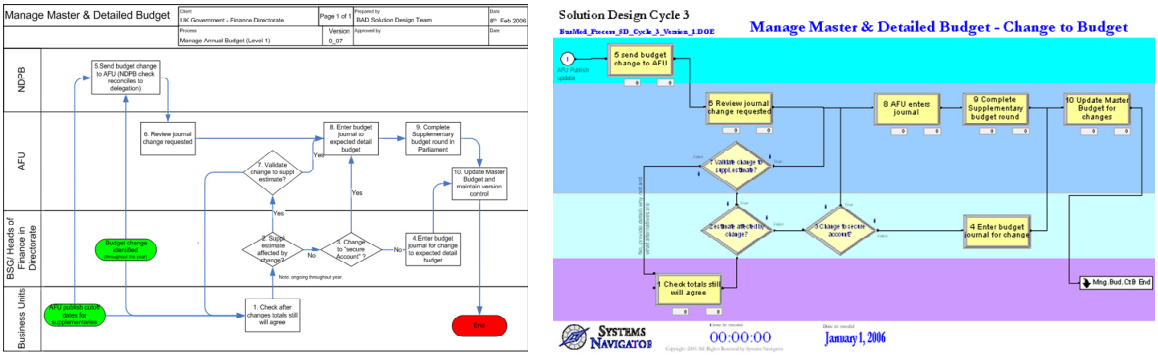


Figure Appendix.3: Part of design (left) and simulation model (right) of cycle 3.

Therefore, a domain specific extension was developed that matched with the VISIO drawings and could represent the work teams easily (Systems Navigator, 2006). Figure Appendix.3 shows at the left hand side the VISIO drawing of a part of the design of cycle 3. At the right hand side is the simulation model instantiated using model constructs of the new domain specific extension.

The use of the domain specific extension resulted in more effective development of the simulation models of the other cycles. The models have been developed in a quarter of the time and the problem owners can compare the simulation models better with the initial design process drawings in VISIO.

Appendix 1.4 Emergency rooms in Hospitals UK

Hospitals in the UK are enforced by the NHS (National Health Service) to provide the appropriate care to patients that enter into the emergency within a certain time limit. The difficulty in providing the appropriate care within the time boundaries is to avoid underutilization of doctors, nurses and spaces and still be capable of handling the variability of arrivals of patients. Discrete event simulation is an increasingly popular tool to provide insight into the quality regarding waiting times a hospital offers.

The processes in hospitals are often modeled by a standard flow of patients that try to seize doctors, nurses and spaces to enforce that they receive the needed care. In reality the doctors determine when a patient is taken care of and doctors will reply differently to requests of patients with different needs and different waiting time. The longer a patient is waiting, the higher its priority will become. The basic model constructs of generic simulation environments are not suited to handle the priority and active doctors without additional allocation mechanisms. We have designed a domain specific extension that consists of simulation building blocks that can handle these priority mechanisms and active doctors (Hay et al, 2006).

Hospitals are forced by regulation to work in the same way. Further doctors have worked for years on standardization of processes to be able to help patients the best way. Therefore, hospitals seem one of the most likely fields to apply a domain specific extension.

The domain specific extension has been used for modeling four different hospitals in the UK. They used the simulation model to analyze the allocation of patients in 2006. The set of simulation building blocks is implemented in the generic simulation environment Arena together with an Excel sheet that allows setting of parameters, creation of doctors and nurses, validate data entry and provide output performance indicators. Figure Appendix.4 shows the parameter setting and ability to create new doctors in the system. Figure Appendix.5 shows the occupancy of the available doctors over time (Hospital Navigator, 2007).

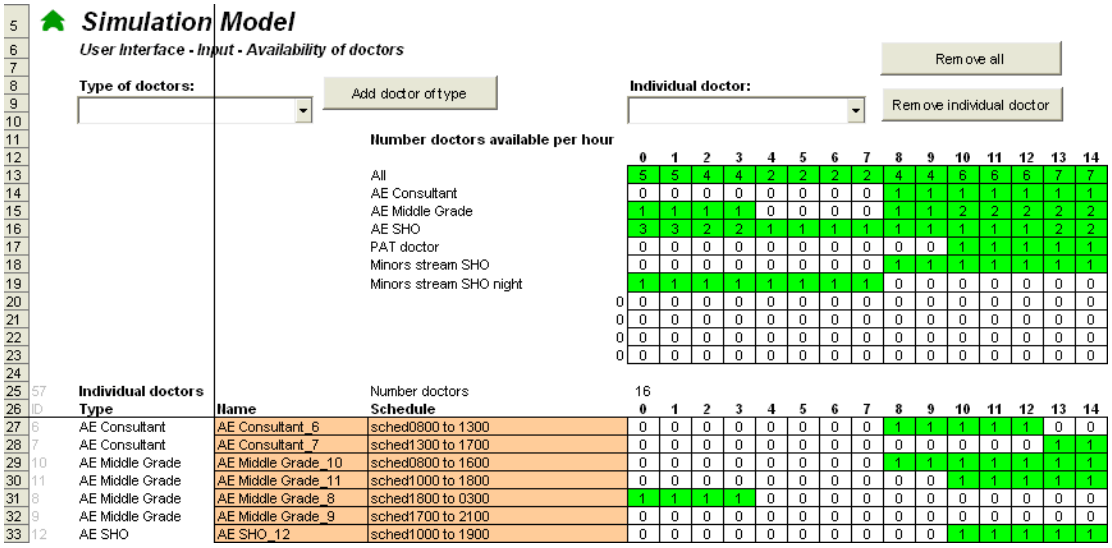


Figure Appendix.4: Set doctor availability in simulation model (Hospital Navigator, 2007 p86)

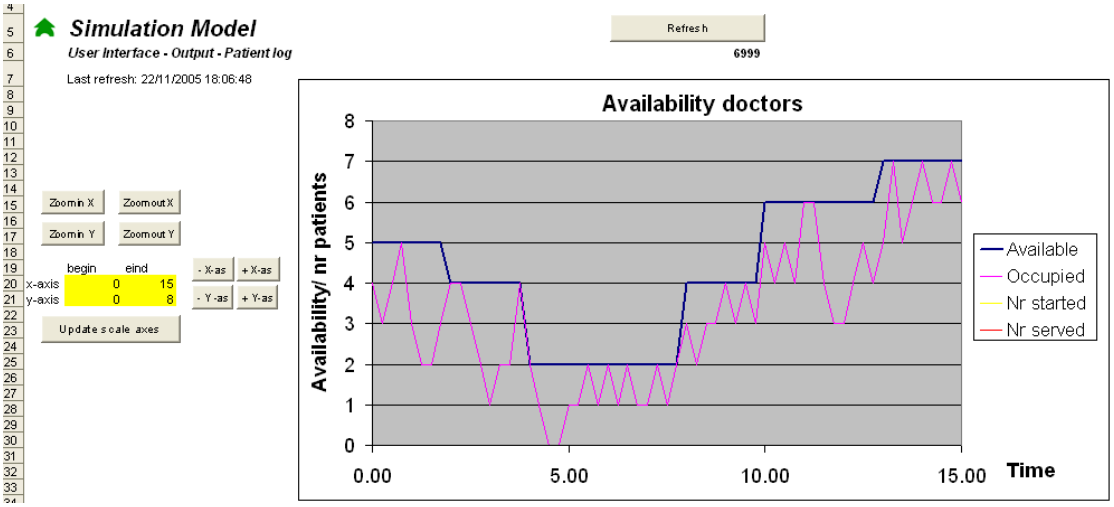


Figure Appendix.5: Output of doctor availability and occupancy during simulation run (Hospital Navigator, 2007, p91)

The designed simulation building blocks have also been implemented in the simulation environment DSOL (Jacobs, 2005). Bajnath and Bani Hashemi (2007) describe how they reused the concept of Hay et al (2006). They

developed a domain specific extension with the DSOL library. A small fictive emergency room was instantiated using the simulation building blocks. The domain specific environment was extended with an interface for setting parameters and analyzing results as part of an applet. Figure Appendix.6 shows this applet running via Internet Explorer (www.hospitalnavigator.com; visited 22-03-2007).

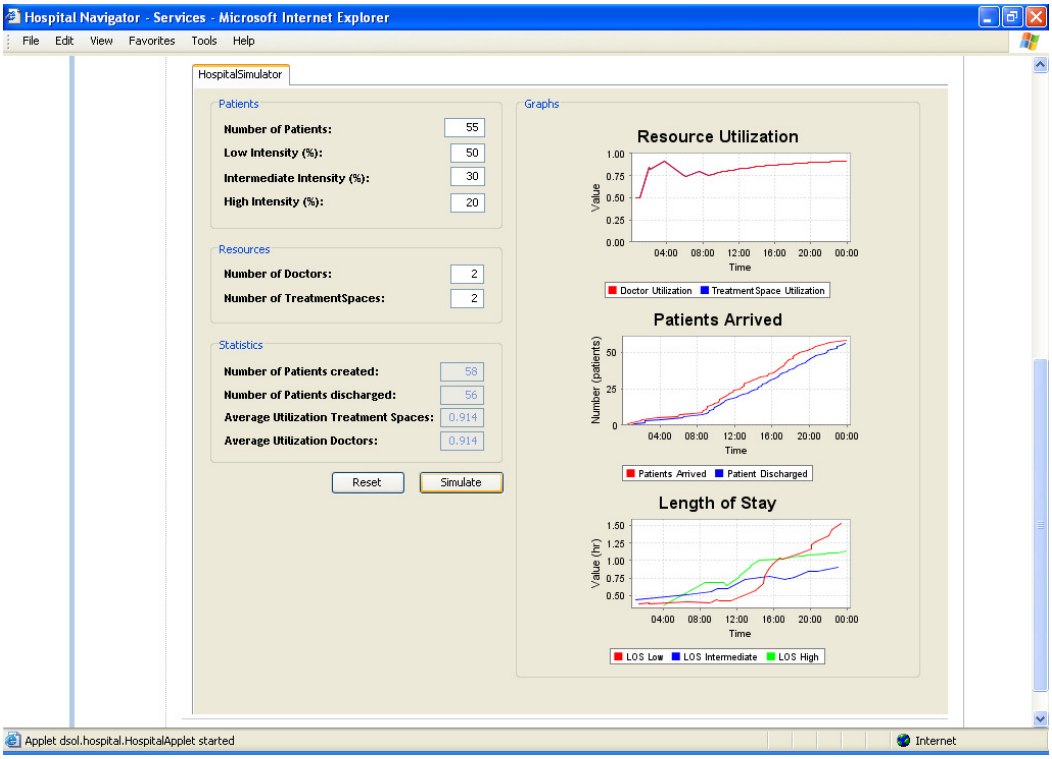


Figure Appendix.6: Applet with interface to DSOL simulation model of hospital

Appendix 1.5 Waterways and vessels

The design of a new canal in France between Paris and Belgium was the trigger of the development for a domain specific extension for the modeling of vessels and waterways. The new canal should consist of 6 to 10 locks, but the size and location between the locks was not known yet. Together with maritime experts of Sogreah we have defined infrastructure and processes of vessels.

This domain specific extension has been the first where the Scenario Navigator software has been applied for full control over the configuration of the simulation model, extension of the simulation model and gather results of the different simulation studies.

A couple of the domain specific building blocks to represent system elements of infrastructure are waterway, bridge, tunnel and lock. The processes defined are the mechanism of priority in a lock or underneath a bridge and reservation of positions in a lock (Systems Navigator, 2005).

The domain specific extension has in addition been applied for simulation studies of the locks near Grave (NL) and an improvement of the canal “Du Rhône à Sète” (FR). These two additional simulation studies required small

extensions to the domain specific extension. A new simulation building block representing two parallel locks has been composed out of existing building block elements for the locks near Grave (NL) and additional building block elements have been developed for the generation of vessels that use the canal “Du Rhône à Sète”.

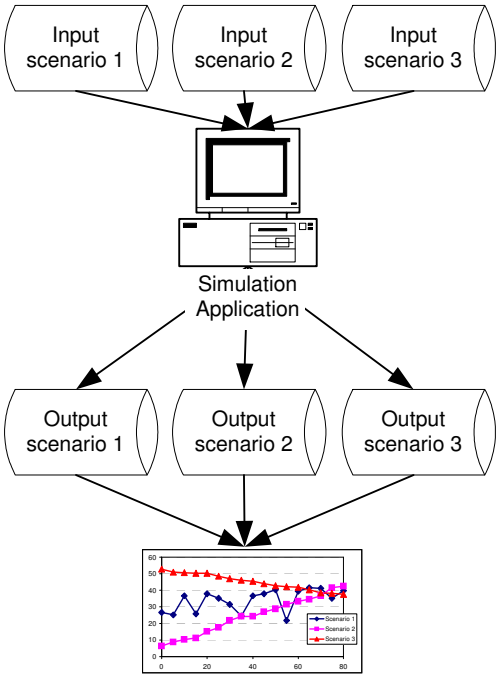


Figure Appendix.7: Effective comparison of simulation experiments of waterways using Scenario Navigator (Valentin et al, 2005a, p1582)

The simulation building blocks have been prepared to be filled with data via the application Scenario Navigator, which enables easy parameterization and optimization for a large set of simulation experiments as well as multiple simultaneous users involved in the simulation studies for the design of the waterways. Figure Appendix.7 shows the use of Scenario Navigator and simulation models developed using the domain specific extension for waterways (Valentin et al, 2005a).

Appendix 1.6 Reorganizing police services to match reaction times

Politicians in the Netherlands have put requirements to the reaction times of police services. A police officer should be able to respond within a certain time frame to an assignment, but also finish the paperwork correctly. Several police departments in the eastern part of the Netherlands have joined forces to be more certain of their capabilities of replying to calls of residents in their area and to be able to reduce the number of individual police officers they need to hire.

A domain specific extension has been developed to represent police officers with their schedules, availability and skills. An interface has been

developed to allocate the correct number of police officers to each individual area and to filter incoming calls from their historical database. The development of the domain specific extension was worthwhile, even though only one simulation study was performed. The adjustments to the simulation model that were enabled by the simulation building blocks and the parameterization instrument were performed much easier than with a simulation model instantiated with generic model constructs. Also the representation of statistics was better thanks to the building block element that collected the statistics of the individual police officers. Figure Appendix.8 shows one of the allocations of police officers to the three sub-areas and the organization of offices instantiated in a simulation model.

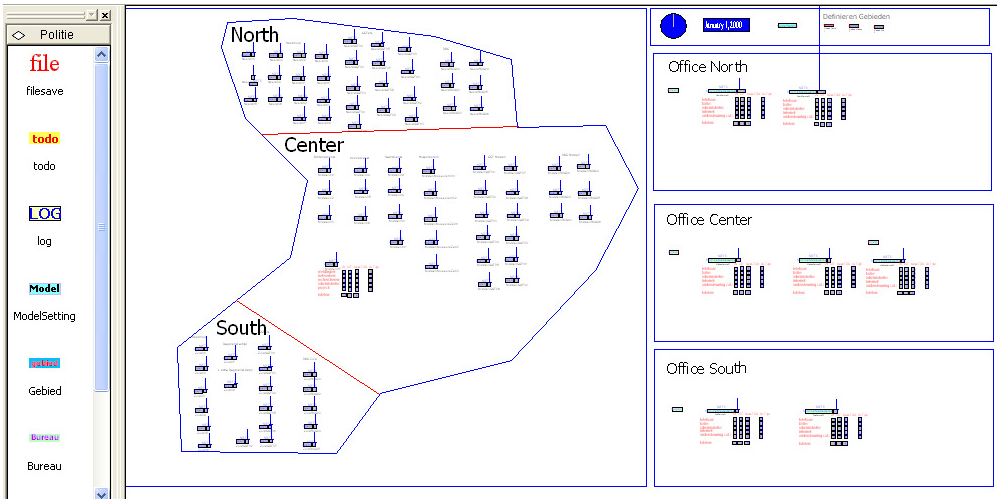


Figure Appendix.8: Simulation model allocation police officers to areas

The experiments that have been performed included applying new technology to provide process improvements and relocating police officers. Thanks to the simulation experiments the police department had insight in the suitability of different configurations and which additional technologies were worth investing (De Vreede et al, 2002)

Appendix 1.7 International banking ABN AMRO

Globalization offers advantages to banks, thanks to companies that perform international business. However, the globalization also results that banks have to deal with more competitors. The ABN AMRO triggered an investigation whether it would be worthwhile to centralize some of their payment activities from offices all over the world to a large dedicated office in the Netherlands. Ayad and Sol (2002) describe issues that the bank has to consider like local regulation and IT-architecture. Ayad and Sol defined together with ABN AMRO several possible allocations of tasks and logistical flows that are required to evaluate

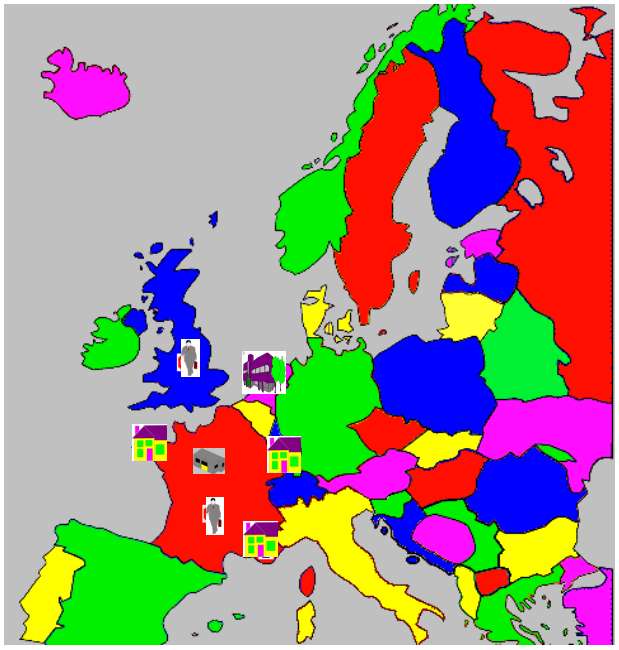


Figure Appendix.9: Visualization interaction local and centralized bank organization (Ayad and Valentin, 2001, p2)

Simulation has been used to evaluate the possible allocations of tasks in the international network of the ABN AMRO. Figure Appendix.9 shows an example simulation model that resembles the map of Europe with local offices in France and London and the interaction with the centralized office in Amsterdam. The figure of a man represents a personal visit of someone to a local brand of the ABN AMRO bank. Instantiating building block elements inside the simulation building blocks of the bank offices enabled allocation of tasks to the local or centralized office (Ayad and Valentin, 2001).

Appendix 1.8 Mail delivery Sandd

Sandd is one of the new companies active at the Dutch mail market. They are specialized in delivery to advertisement material and magazines. Since 2000 the company has been growing fast and captured the second spot of the Dutch post market. The growth has led in 2005 to loss of quality and increases in cost. Sandd has been looking for an instrument to help them in their operational planning and redesign of their activities to be able to cope better with the variability in workload. This instrument should also allow them to predict the future and possible bottlenecks that might arise.

Together with a team of analysts of Boer and Croon Young Executives (www.BCYE.nl) and Sandd we have developed a domain specific extension containing building blocks that represent the processes of post sorting and transportation. The domain specific extension also contains simulation building blocks that represent employees and equipment that is used for sorting. Figure Appendix.10 shows the processes of post sorting in the central hall and the types of equipment used for the sorting.

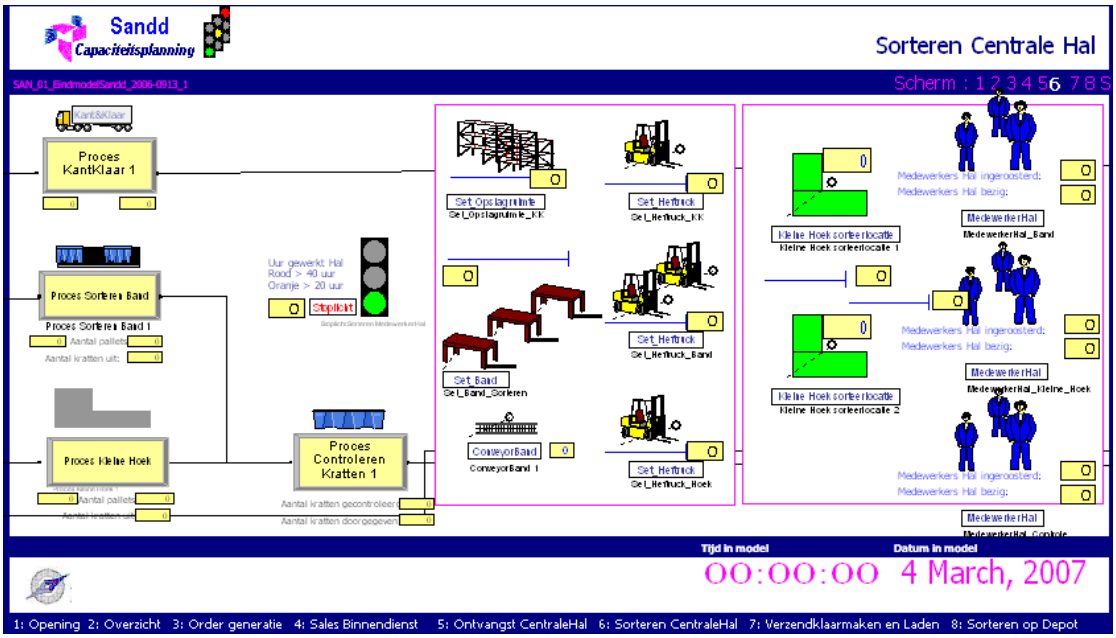


Figure Appendix.10: Post sorting hall of Sandd

The domain specific extension uses Excel sheets for setting parameters of the building blocks and representing output statistics. Scenario Navigator technology is applied to enable that the simulation models can be used by managers in the sales department, the sorting hall, or the decentralized depots to perform experiments with the expected workloads or possible new customers (Riesenkamp et al, 2007).

Summary

Discrete event simulation modeling is for many years already a successful instrument to support problem owners in getting more insight into their problem domain and possible solutions. Eventhough its success, there are also some pitfalls around the use of discrete event simulation. These pitfalls can be summarized around three topics: 1) the model developer has difficulties to handle the unlimited freedom in modeling; 2) model developers need to be experts in multiple areas including programming, statistics and general consultancy; 3) model developers do not speak the language of the problem owner as a model developer cannot specialize as much into one domain. The effect is that simulation studies take much longer then problem owners expect and that the problem owners do not receive the amount of insight requested.

A resolution is found in the use of domain specific extensions to the use of generic simulation environments. Domain specific extensions consist of model construct that represent system elements at the level of abstraction that the problem owner is familiar with. These extensions enable the simulation model developer to create simulation models in a generic simulation environment like Arena or eM-plant, but do not require the model developer to work out all the detailed bits and pieces that are normally part of a simulation model. The model developer does not use model constructs with generic names as ‘queue’ and ‘resource’, nor will the model developer compose the simulation model with processes as ‘wait’ or ‘claim’. The model developer can use model constructs that represent system elements in the level of generalization the problem owner is used to. For example, ‘AGV’, ‘check-in counter’ or ‘ship quay’. As a result the model developer and the problem owner easily speak the same language. Secondly, the model constructs require less technical skills from the model developer and thirdly it reduces the options from the model developer and guides him/her in the direction for composing a simulation model that represents the system of the problem owner.

In literature the use of domain specificity has been seen as an enormous advantage, especially to support the model developer in faster composing the simulation models, but on the other hand the number of domain specific extensions is limited. The same literature also refers to a wide range of risks of using domain specific extensions and discussions with simulation experts confirm the occurrences of these risks in practical case studies.

This research aimed at finding solutions to the risks of using domain specific extensions in discrete event simulation studies. The applied approach has been inductive research via case studies, supported by literature search and laboratory experiments.

Initially two case studies have been performed based on the best knowledge available for the development and use of domain specific extensions in discrete event simulation studies. These case studies dealt with an underground logistics system and passengers at Amsterdam Airport Schiphol and JFK International in New York. In both domains a new domain specific extension has been developed, including a wide range of model constructs that represented system elements that were recognizable parts of the system. With these model constructs several simulation models have been created as part of performed simulation studies. The simulation studies in both domains demonstrated that benefits can be achieved with domain specific extensions that are developed for the domains of AGVs and passengers at airports. However, the case studies also demonstrate that there are a couple of risks that come up when using model constructs from a domain specific extension. We identified new benefits and we had to mitigate new risks in addition to the expected benefits and the predicted risks originating from literature for the use of domain specific extensions in discrete event simulation studies.

Besides the use of domain specific extensions in real life simulation studies we also performed laboratory experiments in which we compared the use of domain specific extensions versus the use of only model constructs of generic simulation environments. The laboratory experiments have been performed with novices (participants without any practical experience in the field of simulation) and experts (participants who were simulation professionals with at least several years experience) in the field of discrete event simulation. In several laboratory experiments the different participants had to make adjustments to existing simulation models to enable new experiments, the participants had to develop simulation models from scratch and the participants had to execute a miniature simulation study. The novice participants that worked with the domain specific extension were better capable to reach the results within the limited time compared to the novices that worked with the generic simulation environment. Further, it was surprising to notice how much difficulties the simulation experts had to work with the model constructs provided by the domain specific extension. The novices were well supported by the domain specificity of the model constructs, while the experts wanted to see the details behind the domain model constructs and felt limited by the domain specific extension.

The observations from the case studies and the laboratory experiments are used to define requirements for a successful domain specific extension. These requirements are translated to a theory that a domain specific extension is more than a collection of model constructs. The contribution of this dissertation provides four elements that help to ensure the expected benefits and mitigate the identified risks. These four elements are:

- Domain specific extension should consist of simulation building blocks and building block elements that match 22 guidelines.
- Additional tools should be available to support data entry, model development and/or output evaluation and comparison.
- The simulation building blocks and building block elements should be well documented and supported by small examples to enable the model developer to feel comfortable with the domain specific extension.
- A design approach needs to be followed that allocates time to the activities for conceptualization and specification.

The concept of simulation building blocks and building block elements are retrieved from research that has been performed in the domain of software engineering. The 22 defined guidelines help to structure simulation building blocks and to improve their reusability by composing the simulation building blocks out of building block elements. Further the guidelines help the developers of simulation building blocks to provide flexible interfaces and user interfaces so model developers can set parameters of the building blocks.

The additional tools automate some of the activities a model developer has to perform in a simulation study. One of the benefits of the use of domain specific extensions has been that statistics is all included in the simulation building block. This inclusion results in a lot of detailed data gathered during the execution of the simulation model, but a model developer cannot handle all this information unless it is structured and summarized. Additional tools help to combine the information and enable the model developer to do faster more experiments that all provide quality insights into the system.

The experts in the laboratory experiments dived into the details of the model constructs of the domain specific extension. They did not have the trust that the model constructs preformed correctly. In the simulation studies performed with the AGVs and at the airports similar observations have been made. Therefore the domain specific extension should provide a lot of documentation and support to enable model developers not only to use the model constructs, but more important to trust the domain specific model constructs.

The domain specific extension is with the introduction of simulation building blocks, the use of additional tools and the documentation and support material much more than just a set of domain specific model constructs developed using a generic simulation environment. Therefore, the approach to design and develop a domain specific extension as it is proposed based on literature studies has been extended, deepened and improved to better support the developers of domain specific extension to follow the guidelines and result in a successful, reusable domain specific extension.

The contribution to the theory of domain specific discrete event simulation (concept of simulation building blocks, additional tools, documentation and support and a design approach) have been applied in a dozen different case

studies. We picked out three that are most representative to demonstrate how the theory is applied, but more important the result of applying the contributed theory in the execution of simulation studies using a domain specific extension of simulation building blocks including additional tools and documentation and support. The three case studies are:

- Information flows for supply chains; especially focus on the use of the design approach and demonstrate the capability to develop the same simulation building blocks independent of generic simulation environment
- Container terminal simulations in 15 minutes; especially focus to the capability of automating simulation model development and output reporting to enable a design for a container terminal to be developed in 15 minutes and provide an extensive documentation of the output of financial and logistical performance indicators
- Nestlé production facilities; especially the use of the simulation building blocks in challenging simulation studies that change scope and require extension of the capabilities of the building blocks that are part of the domain specific extension

The evaluation of these case studies led to the conclusion that domain specific extensions improve the effectiveness of simulation studies, but that the development of such a domain specific extension should follow the guidelines and design approach. Further some additional research is suggested in different areas. We list some possible enhancements to generic simulation environments to better support the development of domain specific extensions. We suggest to investigate the possibilities of new use of simulation models developed using domain specific extensions and we suggest improvements to the design approach to support developers in trade offs to be made during the design of simulation building blocks.

Samenvatting

Simulatie is al jaren een succesvol instrument dat probleemeigenaren ondersteunt in het verkrijgen van inzicht voor mogelijke oplossingen in hun probleemgebied. Ondanks successen rond het gebruik van discreet event simulatie zijn er valkuilen die modelleurs vaak tegen komen. Deze valkuilen kunnen worden samengevat rond 3 onderwerpen: 1) de modelbouwer heeft moeite met de ongelimiteerde vrijheid in de modelbouw; 2) de modelbouwer moet expert zijn in meerdere gebieden, onder andere programmeren, statistiek en algemene consultancy; 3) de modelbouwer spreekt niet de specifieke taal van de probleemeigenaar, want een modelbouwer kan zich niet specialiseren in een domein. Het effect van deze valkuilen die modelbouwers moeten ontwijken is dat simulatiestudies langer duren dan wat probleem-\eigenaren verwachten en dat de probleemeigenaren niet het inzicht krijgen dat ze nodig hebben.

Het gebruik van domeinspecifieke uitbreidingen van bestaande generieke simulatieomgevingen levert hier een oplossing voor. Domeinspecifieke uitbreidingen bestaan uit modelconstructen die een systeem representeren op het abstractie niveau waar de probleemeigenaar gewend is in te werken. Deze uitbreidingen stellen de simuatiemodelbouwer in staat om simulatiemodeleln te bouwen in generieke simulatieomgevingen zoals Arena of eM-Plant, maar de modelbouwers hoeven zich niet te buigen over de technische details die normaal een onderdeel zijn van simulatiemodellen. De modelbouwer gebuikt geen modelconstructuen met generieke namen zoals ‘wachtrij’ en ‘resource’, en ook zal de modelbouwer geen generieke processen in het model opnemen zoals ‘wacht’ of ‘claim’. De modelbouwer kan modelconstructen gebruiken die systeemelementen representeren op het niveau van generalisatie dat de probleemeigenaar gewenst is. Bijvoorbeeld, ‘AGV’, ‘check-in counter’ of ‘scheepskade’. Allereerst kunnen de probleemeigenaar en de modelbouwer dankzij de domeinspecificiteit gebruik maken van dezelfde taal, ten tweede hoeft de modelbouwer minder technische kennis te hebben en ten derde vermindert het gebruik van modelconstructen van domeinspecifieke uitbreidingen de opties van de modelbouwer en stuurt het de modelbouwer bij het bouwen van het simulatiemodel dat het systeem van de probleemeigenaar representeert.

In de literatuur wordt het gebruik van domeinspecificiteit gezien als een groot voordeel, met name om de modelbouwer te ondersteunen om sneller de simulatiemodellen te ontwikkelen, maar aan de andere kant is het gebruik van het aantal domeinspecifieke uitbreidingen beperkt. Dezelfde literatuur verwijst ook naar de diverse risico’s die het gebruik van domeinspecifieke uitbreidingen met zich meebrengen. Deze risico’s zijn is bevestigd in

gesprekken met simulatieexperts die de risico's herkennen in hun eigen simulatiestudies.

Dit onderzoek richt zich op het vinden van oplossingen voor de risico's van het gebruik van domeinspecifieke uitbreidingen in simulatiestudies. In dit onderzoek is gebruik gemaakt van inductief research met behulp van case studies, ondersteunt door literatuuronderzoek en laboratorium-experimenten.

Initieel zijn twee casestudies uitgevoerd op bases van de best beschikbare kennis voor de ontwikkeling en gebruik van domeinspecifieke uitbreidingen in simulatiestudies. Deze casestudies hadden betrekking op een ondergronds logistiek systeem en passagiers op Amsterdam Airport Schiphol en JFK International in New York. In beide domeinen is een nieuwe domeinspecifieke uitbreiding ontwikkeld, inclusief een variëteit aan modelconstructen die herkenbare systeemelementen representeerden. Met deze modelconstructen zijn verscheidene simulatiemodellen gebouwd als onderdeel van de uitgevoerde simulatiestudies. De simulatiestudies in beide domeinen hebben aangetoond dat de verwachte resultaten behaald kunnen worden met domeinspecifieke uitbreidingen die zijn ontwikkeld voor de domeinen van respectievelijk AGVs en passagiers op luchthavens. Echter, de case studies hebben ook getoond dat er risico's zijn bij het gebruik van domeinspecifieke uitbreidingen. We hebben nieuwe risico's en aanvullende voordelen geïdentificeerd in aanvulling op de verwachte voordelen en de voorspelde risico's op basis van de literatuur over het gebruik van domeinspecifieke uitbreidingen in simulatiestudies.

Naast het gebruik van de domeinspecifieke uitbreidingen in simulatiestudies zijn ook laboratorium experimenten gedaan waarin we keken wat het verschil is tussen het gebruik van een domein specifieke uitbreiding en het gebruik van alleen maar een generieke simulatieomgeving. De laboratoriumexperimenten zijn uitgevoerd met beginners (deelnemers zonder enige praktijkervaring met simulatiestudies) en simulatieexperts (deelnemers die minstens enige jaren professioneel simulatiestudies uitvoeren). In de laboratoriumexperimenten moesten de deelnemers aanpassingen maken aan bestaande simulatiemodellen om nieuwe experimenten mogelijk te maken, de deelnemers moesten simulatiemodellen maken vanaf nul en deelnemers moesten een mini simulatiestudie uitvoeren. De beginners die werkte met de domeinspecifieke uitbreiding waren beter in staat om resultaten te verzamelen met de kort beschikbare tijd vergeleken met de beginners die werkten met de generieke simulatieomgevingen. Daarnaast was het verrassend hoe moeilijk de simulatieexperts het hadden om te werken met de modelconstructen van de domeinspecifieke uitbreiding. De beginners voelden zich goed ondersteunt door de domeinspecifieke modelconstructen, terwijl de experts de details achter de domeinspecifieke modelconstructen wilden zien en zich gelimiteerd voelden door het gebruik van de domeinspecifieke uitbreiding.

De observaties van de casestudies en de laboratoriumexperimenten zijn gebruikt voor het samenstellen van eisen voor een succesvolle domeinspecifieke uitbreiding. Deze eisen zijn vertaald naar een theorie dat een domeinspecifieke uitbreiding meer is dan alleen maar een collectie van modelconstructen. De theorie beschreven in deze dissertatie is verdeeld over vier onderwerpen:

- Domeinspecifieke uitbreiding moet bestaan uit simulatiebouwstenen en bouwsteenelementen die 22 richtlijnen volgen.
- Aanvullende applicaties moeten beschikbaar zijn ter ondersteuning van data invoer, modelbouw en/of resultaatevaluatie en resultaatvergelijking.
- De simulatiebouwstenen en bouwsteenelementen moeten uitgebreid beschreven zijn en ondersteund worden door kleine voorbeeldmodellen die de modelbouwer een comfortabel gevoel geven voor gebruik van de domeinspecifieke uitbreiding.
- Een ontwerpaanpak moet gevolgd worden die veel tijd besteed aan de activiteiten voor conceptualisatie en specificatie.

Het concept van simulatiebouwsteneen en bouwsteenelementen is gebaseerd op onderzoek in het gebied van software engineering. De 22 gedefinieerde richtlijnen helpen om simulatiebouwstenen te structeren en verbeteren de herbruikbaarheid door het samenstellen van simulatiebouwstenen op basis van bouwsteenelementen. Daarnaast helpen de richtlijnen de ontwikkelaars van simulatiebouwstenen in het maken van een flexibele interface en gebruikersinterfaces zodat modelbouwers parameters van de bouwstenen kunnen instellen.

De aanvullende applicaties automatiseren enkele van de activiteiten die een modelbouwer moet uitvoeren tijdens een simulatiestudie. Een van de voordelen van het gebruik van domeinspecifieke uitbreidingen is dat statistieken al onderdeel zijn van de simulatiebouwstenen. Dat betekent dat automatisch veel gedetailleerde informatie beschikbaar is na de uitvoer van een simulatiemodel, maar een modelbouwer kan niets met al deze informatie tenzij het wordt gestructureerd en samengevat. Aanvullende applicaties helpen om deze informatie te combineren en ondersteunen de modelbouwer om sneller experimenten uit te voeren die inzichten geven in het systeem.

De expertdeelnemers van de laboratoriumexperimenten doken in de details van de modelconstructen van de domeinspecifieke uitbreiding. Zij hadden niet het vertrouwen dat de modelconstructen correct werkten. Vergelijkbare observaties zijn gemaakt in de simulatiestudies met de AGVs en de luchthavens. Om dit te voorkomen moet de domeinspecifieke uitbreiding veel documentatie en ondersteuning bieden aan de modelbouwers, niet alleen over het gebruik van de modelconstructen, maar nog belangrijker over het vertrouwen dat de modelbouwer mag hebben in de modelconstructen.

De domeinspecifieke uitbreiding is met de introductie van de simulatie-bouwstenen, het gebruik van aanvullende applicaties en het documentatie- en supportmateriaal veel meer dan alleen maar een set van domeinspecifieke modelconstructen ontwikkelt in een generieke simulatieomgeving. Daarom is de ontwerpaanpak die is geïntroduceerd op basis van literatuuronderzoek uitgebreid, uitgediept en verbreed om de ontwikkelaars van domeinspecifieke uitbreidingen beter te ondersteunen om de simulatie bouwsteen richtlijnen te kunnen volgen en te kunnen resulteren in een herbruikbare domeinspecifieke uitbreiding.

De bijdrage aan de theorie van domeinspecifieke uitbreidingen (concept van simulatie bouwstenen, aanvullende applicaties, documentatie en support en een ontwerpaanpak) is toegepast in een dozijn verschillende casestudies. We hebben drie casestudies geselecteerd die het meest representatief zijn om te demonstreren hoe de theorie is toegepast en om te laten zien wat het resultaat is als deze theorie wordt toegepast in simulatiestudies met behulp van domeinspecifieke uitbreidingen van simulatiebouwstenen inclusief aanvullende applicaties en documentatie en support. De drie case studies zijn:

- Informatie voorziening voor supply chains; met name focus op het gebruik van de ontwerpaanpak en aantonen dat het mogelijk is om simulatiebouwstenen te maken ongeacht de generieke simulatieomgeving waarin de bouwstenen worden geïmplementeerd.
- Containerterminalsimulatie in 15 minuten; met name focus op de mogelijkheid om door automatische simulatiemodel ontwikkeling en automatische aggregatie van simulatiemodel data een containerterminal te ontwerpen in 15 minuten en uitgebreid de resultaten te documenteren van de financiële en logistieke kengetallen.
- Nestlé productiefaciliteiten; met name het gebruik van simulatiebouwstenen in simulatiestudies die door wijzigingen resulteren in uitbreidingen van de bouwstenen die onderdeel zijn van de domein-specifieke uitbreiding.

De evaluatie van deze casestudies heeft geleid tot de conclusie dat domein-specifieke uitbreidingen de effectiviteit van simulatiestudies verbeteren, maar dat de ontwikkeling van deze domeinspecifieke uitbreiding wel gedaan moet worden volgens de ontwerpaanpak en rekening houdend met de richtlijnen. Er is aanvullend onderzoek geïdentificeerd ijn verschillende onderwerpen. We adviseren verbeteringen voor de generieke simulatie-omgevingen met betrekking tot ontwikkeling van domein-specifieke uitbreidingen. We adviseren ook om te onderzoeken naar alternatief gebruik van simulatiemodellen ontwikkeld met domeinspecifieke uitbreidingen en we adviseren onderzoek naar verbeteringen van de ontwerpaanpak om ontwikkelaars van simulatiebouwstenen beter te ondersteunen in het maken van ontwerpkeuzes.

Curriculum Vitae

Edwin Valentin was born in Delft at 21st of September 1976. He has been raised in Naaldwijk and joined there the HAVO. After graduation in 1993 he joined the HES (Hogeschool voor Economische Studies; College for economic studies) and graduated in 1997 in the direction of Logistics and Economics after 4 years of study including a semester at the University of Westminster in London.

Edwin joined in 1997 the consultancy branch of Incontrol Business Engineering and produced his first simulation models with Arena and eM-Plant for customers such as Unimills, Prorail and CTT. In 2000 Edwin switched to the Delft University of Technology, Faculty of Technology, Policy and Management, section Systems Engineering for a PhD-research position as part of the BETADE research project group. In 4.5 years Edwin performed over 15 different projects, delivered more than a dozen articles for conferences, book chapters and journals and actively participated in the education program by giving classes to students ranging from first years to last year master students.

In 2004 Edwin shifted his career to Systems Navigator where he build on the development of a consultant department by performing simulation projects for (international) companies such as Total, Nestlé, Sogreah, Sandd and Samskip. Almost all his projects included the development of a domain specific extension and some of the projects are included in this dissertation as example.

Currently Edwin is working for the consultancy firm Accenture where he specializes in delivery management for large scale projects. His most recent projects are a data migration project and implementation of Oracle Retail.