Delft University of Technology,
Faculty of Technology, Policy and Management,
Department of Systems Engineering

MASTER THESIS

simulation & distributed visualization
within an architecture for flexible, real-time
monitoring & control of devices

Submitted by:
Niels Lang
Telephone: +31-628596969
E-mail: N.A.Lang@tbd.tudelft.nl

Advisors:
Prof. dr. H.G. Sol (TU Delft)
Dr. ir. A. Verbraeck (TU Delft)
Dr. P.W.G. Bots (TU Delft)
Drs. L. Bonebakker (Sun Microsystems)

’s-Gravenhage, fall 2001
**Preface**

This report is one of the results of a six month project carried out at Sun Microsystems's iForce Ready Center in Menlo Park, CA, USA. This project took place in the iForce's Trading exchanges group and was funded by Sun Microsystems. The project resulted from the cooperation between Delft University of Technology (TU Delft) and Sun Microsystems (Sun), and was carried out in the first half of 2001 by three students of the TU Delft's faculty of Technology, Policy and Management.

The initial objective for the project was to develop a visualization system to leverage iForce's ability to provide customers with more insight in the benefits that Sun's systems could realize for their (E-)business. This resulted in the development of a general architecture for distributed visualization & control, that is introduced in a general introduction on this project ([HissinkMuller01]). The report you are reading now presents the research on the field of simulation and visualization conducted by the author during this project.

The project was coached by prof. dr. H.G. Sol, dr. ir. A. Verbraeck and dr. P.W.G. Bots on behalf of the faculty of Technology, Policy and Management and by drs. L. Bonebakker on behalf of the iForce Ready Center at Menlo Park. The following paragraphs introduce Sun Microsystems, the iForce initiative, the Trading exchanges group and the Systems Engineering Group at Delft University of Technology.

**Sun Microsystems Inc.**

Sun Microsystems, a $20 billion company founded 18 years ago with offices in 170 countries, provides end-to-end solutions for doing business in the network age. Sun's products and services cover all the aspects of ICT, from reliable and scalable high-end enterprise servers, to the platform independent and object-oriented Java programming language.

**iForce initiative**

The iForce initiative within Sun as a part of Global Technical Operations aims to provide solution blueprints for mission critical applications that *Run-On-Sun* to deliver sustained business advantages for an enterprise.

The blueprints are not only designed, but also tested for the customer by means of the Proof of Concept trajectory. During the Proof of Concept trajectory the critical parts of the customer’s solution will actually be built to be able to ‘prove’ the performance of the proposed blueprint.

The iForce Ready Centers consist of various groups with different focus. There are product-specific groups like the SAP-group, the Oracle/PeopleSoft and the iPlanet-group, as well as applica-
tion-specific groups like the Supply Chain Management group and the Trade exchanges group, where this project was conducted.

**iForce Ready Center Menlo Park - Trade exchanges group**

The Trade exchanges group within iForce focuses on large e-Business problems that involve trade exchanges, also known as virtual marketplaces. Many Fortune 100 enterprises coming to the lab get Sun’s help on realizing maximum benefit from the opportunities today’s networked digital economy offers. Because of the complexity in technology involved and the fact that trade exchanges are such a new subject, very specific expertise is necessary to get an E-business system that involves a trade exchange working. In the iForce lab proof of concept is established for these complex cases.

**Systems Engineering Group at Delft University of Technology**

Founded in 1864, Delft University of Technology is the oldest, largest, and most comprehensive technical university in the Netherlands. With over 13,000 students and 2,100 scientists (including 200 full-time professors), it is an establishment of both national importance and significant international standing. Renowned for its high standard of education and research, the University collaborates with other educational establishments and research institutes, both in the Netherlands and overseas. It also enjoys partnerships with governments, branch organizations, numerous firms, the industry, and companies from the small and medium business sectors.

Systems Engineering (SE) is a research group within the faculty of Technology, Policy and Management (TPM). As such its main strengths are in multidisciplinary projects. Central research issues for TPM are the process of problem analysis & solution and complex design trajectories. One of the research directions for SE is ICT, focusing on E-commerce applications in different organizational settings. Group technology and simulation tools are being developed, tested and used within SE to conduct this research. A current research theme, in which these tools are used, is the modeling of complex systems, including their dynamics and control using component based architectures.

**About this report**

This report presents the research conducted by the author during the project described above. A general introduction on this project and the resulting architecture is presented in [HISSINKMULLER01]. In that report, the three separate reports, each one produced by one of the project members, are also introduced.

The conducted research described in this report has focused on the way *visualization & control* and *simulation* contribute to the objectives of the developed architecture. The way simulation
(and in general: any device) are connected to the architecture, by use of \textit{wrapper components} will also be discussed in this report. These three fields of research have been developed (and implemented) as three \textit{subsystems}\footnote{See [HISSINKMULLER01] for a short introduction in systems thinking and engineering.} of the developed architecture.

The Introduction section discusses in more detail the relationship between the researched subsystems and the general architecture. The relevant subsystems are discussed in the subsequent chapters. Detailed information and source code of the reference implementations built for the subsystems can be found in the accompanying appendices and CD-ROM.

The author would like to thank Sun Microsystems for the accommodation and support offered during the full run of the project.

Niels Lang
Abstract

This thesis presents the results of the research conducted at Sun Microsystem’s iForce Ready Center in Menlo Park, CA, on the extension of a basic architecture for remote device monitoring, aimed at leveraging the architecture’s usability as a tool to support decisionmaking.

This basic architecture has initially been developed to meet the problem faced by one of iForce’s customers: a large oil-company owning refineries, storage facilities and about 23,000 gasstations all over the USA.

Due to increasing pressure from the competition, its management is under pressure to cut costs while maintaining market share. In reflection of their cost structure, they felt that potentially significant cost reductions could be achieved by optimizing their operational and support processes, like the supplying and selling of gasoline. However, the management felt that actors within the company were prevented to realize such cost reductions, because relevant and timely information was not available for them. They had also observed that many of the information needed is potentially available in all the devices involved with the company’s operational and support processes.

Sun, recognizing the fact that many more companies may exist that make costs because relevant business information remains locked in devices, took up this challenge to develop an architecture to unlock information in these devices in order to achieve a situation in which the right information gets to the right actor at the right time.

A basic architecture, self-managing thus low-cost, that makes information from devices in the network accessible, in a real-time and actor-specific way, has indeed been developed and is introduced in this report. The focus of the research presented in this report, however, is pointed at extensions of this basic architecture, in order to increase its usability as a tool for decisionmaking with regard to the company’s operational and support processes.

Based on the case just described and theory on decisionmaking, two high potential extensions for the architecture have been identified:

- **Information visualization.** In a company consisting of a distribution network of about 23,000 gasstations, over 50 gasoline storage facilities and about 10 refineries, an enormous amount of information on the company’s operational processes may potentially be generated. Theory on visualization and decisionmaking suggests that visualization of information in its context significantly improves the effectiveness of the information communication with a human actor. It may therefore improve a human’s ability to process and filter this huge information flow, improving his or her decisionmaking capacity.

- **System prediction.** Optimization of processes may require structural changes in the way these processes are designed or controlled. However, the impact of such changes on the process per-
formance, is often hard to predict analytically, given the complexity and dynamics of the process. Instead, simulation models may be used that indeed simulate the complexity and dynamics of the system considered relevant.

In addition, research has been conducted on an operational extension of the architecture, namely the components providing the actual communication between the architecture and the (real or simulated) devices. This is because this proved to be a required connection to test the feasibility of simulation and visualization in the basic architecture.

The approach of this research has been to research the feasibility of implementing the extensions just presented, given the recent technological developments in the field of object-oriented, platform independent and network-enabled programming environments. However, in order to do so, a clearer focus was needed on the actual requirements to be met by these architecture extensions. Therefore this research presents the functional and technical requirements identified and design choices available to meet them. In addition, the lab implementations that have been developed during the project for all three extensions are presented and evaluated. The lab implementations show that is is indeed feasible to extend the basic architecture with visualization and simulation, increasing its usability as a tool for decisionmaking.
Contents

1 Introduction of the basic architecture ................................................................. 1
   1.1 E-Gas case ........................................................................................................ 1
       1.1.1 Company description .............................................................................. 1
       1.1.2 Business processes at a gas station ....................................................... 3
       1.1.3 Problem for E-Gas Station ................................................................... 4
       1.1.4 Problem summary ............................................................................... 5
   1.2 Conceptualization of an architecture ............................................................ 6
       1.2.1 A traditional solution: ERP ................................................................. 6
       1.2.2 The carwash example ......................................................................... 6
       1.2.3 Conceptualization of the architecture ............................................... 7

2 Introduction of the research ............................................................................. 15
   2.1 The architecture extensions introduced ...................................................... 15
       2.1.1 Different contexts, different decisionmaking ....................................... 16
       2.1.2 The architecture extensions identified for support of decisionmaking ....... 18
       2.1.3 The framework extension in the decisionmaking model ...................... 20
   2.2 Design methodology .................................................................................. 22
       2.2.1 Way of thinking ............................................................................... 22
       2.2.2 Way of working .............................................................................. 23
       2.2.3 Way of modelling ........................................................................... 23
       2.2.4 Way of controlling .......................................................................... 24
       2.2.5 Way of working .............................................................................. 24
   2.3 Research question ...................................................................................... 24
   2.4 Structure of the report .............................................................................. 25

3 The Visualization & Control system ................................................................. 28

   Intermezzo 1: Visualization concepts ............................................................... 29
   3.1 Design requirements ................................................................................ 32
       3.1.1 Actor requirements for demonstration .............................................. 32
       3.1.2 Actor requirements for decision-making .......................................... 36
       3.1.3 Technical Requirements .................................................................. 45
3.1.4 Design choices .................................................................................................................................... 48
3.2 The lab implementation .......................................................................................................................... 52

Intermezzo 2: Java introduced .................................................................................................................. 53
3.2.1 The Topic implementation .................................................................................................................. 54
3.2.2 Some Java concepts introduced ......................................................................................................... 55
3.2.3 The visualization Interface .................................................................................................................. 61
3.2.4 Visualization components developed ................................................................................................. 64
3.2.5 Delivering remote objects .................................................................................................................. 67
3.2.6 Discussion on not yet implemented features ....................................................................................... 70
3.3 Conclusions & recommendations ......................................................................................................... 74
3.3.1 The research question addressed ....................................................................................................... 74
3.3.2 Recommendations for further research and development ................................................................. 78

4  Simulation in the architecture ................................................................................................................. 81

Intermezzo 3: simulation defined ............................................................................................................... 81
4.1 Design requirements ............................................................................................................................... 83
4.1.1 Actor requirements ............................................................................................................................... 83
4.1.2 Technical Requirements ....................................................................................................................... 92
4.1.3 Design choices .................................................................................................................................... 96
4.2 The lab implementation .......................................................................................................................... 100
4.2.1 Components used between simulation and virtual device ................................................................. 101
4.2.2 Event based communication ............................................................................................................... 103

Intermezzo 4: Exchanging data between DLLs .......................................................................................... 104
4.2.3 Simulating remote events .................................................................................................................... 105
4.2.4 Features not included in the reference implementation ...................................................................... 108
4.3 Conclusions & recommendations ......................................................................................................... 109
4.3.1 The research question addressed ....................................................................................................... 109
4.3.2 Recommendations for further research and development ................................................................. 112

5  Wrapping devices ..................................................................................................................................... 114
5.1 Requirements ........................................................................................................................................ 115
5.1.1 Actor requirement: low cost wrapper development ............................................................................ 115
5.1.2 Technical requirements ....................................................................................................................... 117
5.2 Design choices ....................................................................................................................................... 118
5.2.1 Choice of runtime environments.................................................................118
5.2.2 Data complexity level......................................................................................119
5.2.3 Monolithic vs. piped approach........................................................................120
5.2.4 Local vs. remote wrapper components........................................................121
5.3 Lab implementation...........................................................................................124
5.4 Conclusions and recommendations...............................................................126
  5.4.1 The research question addressed..............................................................126
  5.4.2 Recommendations for further research and development........................127

6 Conclusions & recommendations.........................................................................129
  6.1 Recommendations..........................................................................................130
    6.1.1 Standardization..........................................................................................130
    6.1.2 Development of modeling components..................................................130
    6.1.3 Development of browsing capability.......................................................130
    6.1.4 Real case performance..............................................................................131

Intermezzo 5: Current Java object query mechanisms.................................vii
List of figures

Illustration 1.1.1: eGas supply chain.................................................................2
Illustration 1.2.1: Carwash example...................................................................7
Illustration 1.2.2: Actors have an interest in information.................................8
Illustration 1.2.3: Devices have a virtual representation in the network.........8
Illustration 1.2.4: Actors have a virtual representation in the network..........9
Illustration 1.2.5: Information extracting filters deliver relevant events to virtual actors..........................................................10
Illustration 1.2.6: Access to system functionality is restricted by an administrator..........................................................10
Illustration 1.2.7: An actor interacts with system functionality through a business rule..........................................................11
Illustration 1.2.8: Various GUI-panels allow actors to receive events anytime, anywhere..........................................................12
Illustration 1.2.9: E-Gas Station instantiation....................................................13
Illustration 1.2.10: Architecture instantiation for E-Gas Station example...............14
Illustration 2.1.1: The theoretical problem solving cycle.................................17
Illustration 2.1.2: Information flows..................................................................18
Illustration 2.1.3: Visualization in the architecture..........................................19
Illustration 2.1.4: Simulation in the architecture..............................................19
Illustration 2.1.5: Wrapper in the architecture..................................................20
Illustration 2.1.6: Phases in the decisionmaking process..................................21
Illustration 2.1.7: Transition from reality to simulation....................................21
Illustration 2.2.1: Way-of framework...............................................................22
Illustration 2.4.1: Structure of this report........................................................26
Illustration 3.1: Visualization & control in the architecture..........................28
Illustration 3.2: Functions of a dynamic visualization system............................30
Illustration 3.3: The information navigation process.......................................32
Illustration 3.1.1: Sample configuration..........................................................33
Illustration 3.1.2: Visualization for demonstrating the performance of a server..34
Illustration 3.1.3: 'Silent' visualization..............................................................38
Illustration 3.1.4: Structural overview of a gasstation......................................38
Illustration 3.1.5: Example of use of different contexts....................................39
Illustration 3.1.6: Transaction phases occurring in different realities...............40
Illustration 3.1.7: Physical connection between devices....................................41
Illustration 3.1.8: A model component in measurement mode........................42
Illustration 3.1.9: A model component in prediction mode .......................................................... 43
Illustration 3.1.10: An active control component ........................................................................ 44
Illustration 3.1.11: UML diagram of three sample event models ................................................. 49
Illustration 3.1.12: Two contexts for the model component ........................................................ 50
Illustration 3.1.13: Virtual connection component ................................................................. 52
Illustration 3.2.1: Implementation aspects .................................................................................. 52
Illustration 3.2.2: Topic matching .............................................................................................. 55
Illustration 3.2.3: Java used in the V&C system ........................................................................ 56
Illustration 3.2.4: Encapsulation of fields .................................................................................. 58
Illustration 3.2.5: Java graphic components .............................................................................. 60
Illustration 3.2.6: Event delivery in the visual component .......................................................... 62
Illustration 3.2.7: Class AnimationComponent ......................................................................... 65
Illustration 3.2.8: A typical animation panel ............................................................................. 65
Illustration 3.2.9: StateChange component ............................................................................. 66
Illustration 3.2.10: Level change component .......................................................................... 67
Illustration 3.2.11: The panel generation process ..................................................................... 68
Illustration 3.2.12: Compents for distributed V&C component delivery ..................................... 68
Illustration 3.2.13: The Topic select tool ................................................................................... 69
Illustration 3.2.14: Structure and flow visualization .................................................................... 71
Illustration 4.1: Simulation in the architecture ........................................................................... 81
Illustration 4.1.1: Internal vs. external simulation ..................................................................... 85
Illustration 4.1.2: A simple gaming simulation ...................................................................... 87
Illustration 4.1.3: Distributed simulation ................................................................................ 88
Illustration 4.1.4: Control system layout & soft commissioning .................................................. 89
Illustration 4.1.5: Reality in the loop ......................................................................................... 90
Illustration 4.1.6: Concurrent connections .............................................................................. 97
Illustration 4.1.7: Approach A: Single event handling mechanism ........................................... 98
Illustration 4.1.8: Approach B: Two event handling processes ................................................ 99
Illustration 4.2.1: Simulation - Virtual Device Interface ............................................................... 101
Illustration 4.2.2: UML diagram of events ................................................................................ 103
Illustration 4.2.3: Remote event handling in Arena .................................................................. 106
Illustration 4.2.4: Remote State Change block ........................................................................ 107
Illustration 4.2.5: Remote property change block ................................................................... 107
Illustration 5.1: The wrapper subsystem................................................................. 114
Illustration 5.2: Wrapper in detail........................................................................... 114
Illustration 5.2.1: Piped components...................................................................... 120
Illustration 5.2.2: Two wrapper designs for a sample real device........................ 121
Illustration 5.2.3: Several distribution policies...................................................... 122
Illustration 5.2.4: A separate wrapper connection................................................ 123
Illustration 5.3.1: Lab implementation of a Corba wrapper................................. 124
Illustration 5.3.2: UML overview wrapper implementation.................................. 125
Illustration 6.1.1: Simple control system................................................................. iv
Illustration 6.1.2: Information flows........................................................................ vi
Illustration 6.1.3: Equal Marshalled Objects........................................................... vii
Illustration 6.1.4: The Matching process illustrated................................................ viii
Illustration 6.1.5: The ‘Match’ process illustrated................................................... x
Illustration 6.1.6: UML diagram of the Topic classes........................................... x
Illustration 6.1.7: The dynamic object editor in action......................................... xiv
Illustration 6.1.8: point property introspection..................................................... xv
Illustration 6.1.9: The collection editor................................................................. xv
Illustration 6.1.10: A federation overview.............................................................. xviii
Illustration 6.1.11: Example interaction model...................................................... xix
Illustration 6.1.12: Example object model............................................................. xix
Illustration 6.1.13: Service categories defined for the HLA interface................... xxi
Illustration 6.1.14: A region in a namespace spanned by 3 dimensions................ xxiv
Illustration 6.1.15: Sample RTI implementation..................................................... xxvi
Illustration 6.1.16: Sequence diagram of event broadcast...................................... xxviii
Illustration 6.1.17: Sequence diagram of poll for external events......................... xxix
Illustration 6.1.18: Event types used in the architecture......................................... xxxii
Illustration 6.1.19: Screenshot of variables in a wrapped Arena simulation............ xxxiii
1

Introduction of the basic architecture

This chapter will introduce the basic architecture for flexible, real-time monitoring & control of devices. The next section introduces the case of a major player in the oil-business that came to Sun’s iForce Ready Center because its management felt the company is losing money due to inefficiency of its business processes and downtime of services, which case has been the basic starting point for the development of the architecture. The case introduces the typical company and problems that the architecture should support.

After that, the basic architecture developed will be introduced, as a tool for remote monitoring and control of various types of devices, so information can be used in various ways to improve business efficiency and levels of service. Further details on the concepts of the basic architecture can be found in [HISSINKMULLER02]. The components the architecture consists of, have been especially developed during the project, in order to develop the architecture as a self-managing network. Details on these basic components can be found in [JACOBS01].

1.1 E-Gas case

Currently, many actors throughout the company’s business chains lack the information needed to optimize the primary and support processes which, as an internal report turned out, could significantly reduce costs and improve competitiveness. Sun’s iForce Ready Center management feels the problems found in the E-Gas business case are found in many large organizations.

This chapter starts with a section containing background information on the company, describes the company’s relevant business processes from the perspective of a single gas station and introduces the company’s problem. This chapter concludes with a short summary of the problem for E-Gas, the goal of this thesis and the research questions.

1.1.1 Company description

E-Gas\(^2\) describes the case of a company, an alliance between two major oil-companies, cooperating in the field of exploration, production and sale of oil-products and services throughout the US. Production facilities in the US cover 8 refineries, which refine over 1.3 million barrels of oil

\(^2\) Throughout this thesis E-Gas Station refers to both the business case and company.
per day. The alliance owns 29,000 miles of pipelines and distributes its oil-products over 23,000 gas stations. In the year 2000 the alliance's total revenue was US$ 69 billion, with a market share of 14.5%. The alliance employs 13,000 people, working from exploration to the sales of gas and services.

Illustration 1.1.1 depicts a typical representation of the E-Gas supply chain, showing how crude oil is won in exploration, refined at one of the refineries and eventually brought to one of 23,000 gas stations all over the US. As indicated by the actors on the picture, a wide range of different internal and external parties are involved in the process of exploration, production, distribution and sale of oil-products. From top-management all the way down to maintenance contractors and people responsible for delivery of new candy-bars to any of the many gas station locations, actors need information to support their decision making.

In over 23,000 gas stations in the US, not all company-owned, money is made in sales of gas and services, such as food and beverages and carwash-services. The operational margin on the sale of gas is relatively low, but because of the huge volume of the market, still a considerable amount of money is made. The margin on the sale of services and non-gasoline products, such as food, beverages and carwash services, is considerably higher, however, this sale is much smaller in volume.

Since substantial amount the costs are made and all revenues are generated close to the end-cus-

---

3 Information gathered from one of the the company's websites.
tomor, the next section gives an overview of the most important downstream⁴ business processes in the company, zooming in on a single gas station, since one gas station is representative for each of the 23,000 throughout the entire US.

1.1.2 Business processes at a gas station

From the perspective of one gas station, several processes and devices are relevant for the business case. The processes and devices described in this section are an abstraction of the real business situation, since not every gas station is identical in size and services offered.

At first it might seem, that the processes which are most important to the gas company are those which involve the sale of gas and services, such as the sale of carwash-services and non-oil products. Although these are the business processes where the actual money is made, the sale of gas and services halts soon if no supplies are delivered or broken devices are not repaired.

Several business processes are relevant, since they are the primary, in the sense that reflect where customers bring in their money:

- sale of gas
- sale of food and beverages
- sale of carwash-services

For the support of the primary processes other processes are important also. Support processes are, for example:

- delivery of gas to the gas station
- delivery of supplies and products to a gas station, e.g. soap for the carwash-installation
- maintenance and repair of gas station infrastructure

Over the entire US several hundreds of different suppliers and maintenance contractors are responsible for the support processes just mentioned. Responsible actors want to optimize these processes, reduce costs and leverage revenue. They need information to do that.

When looking at the business processes just described in more detail, it becomes clear that with all of the processes devices are involved, either because at these devices the customer directly interacts with the system, or because they support or enable the primary processes. The following list identifies some of the devices which are relevant in the business processes at a gas station:

---

⁴ downstream processes are all the business processes regarding the sale of end-products and services, compared to the upstream processes which are involved with the exploration and production of oil-products.
• gas pumps
• underground pump
• point of sales terminal
• carwash installation
• refrigerator inside the shop

From the description of the most important processes and devices it is obvious that with every business process, devices play a key part. Information available in the devices contains vital steering information for actors in the business environment, with regard to business processes. As an example, a description of various types of information available within a carwash-installation is given:
• soap level
• wax level
• water pressure level
• air pressure level

An example of an actor using information from this device is the following:

**Example** A gas station clerk working behind the counter of the gas station needs to act whenever the soap level drops below a certain threshold. If so, he/she needs to refill the soap from the stock which is available in the gas station.

To support his business process of refilling the soap and keeping the carwash operational, the clerk working behind the counter in the gas station would like receive the right information soap went below a certain threshold level at the right time (the moment the event occurred).

Currently, the only way for the clerk to know the soap level is to physically walk up to the carwash-installation.

**1.1.3 Problem for E-Gas Station**

Due to various external factors, such as the economic downturn and the strong competition, the company’s profit margin is under pressure.

In order to boost overall profitability, company management is looking for ways to cut costs, while at the same time increasing the company’s overall levels of service. Currently, the company
is under the assumption that several of the business processes do not perform at high levels of efficiency and that the right steering-information is lacking to optimize these processes. Apart from that, company management thinks that information available in devices can be used (in an aggregated form) to level the information asymmetry between maintenance contractors and the E-Gas company. As contract-terms for maintenance and delivery of supplies and getting ever shorter, information regarding these processes would allow the company to re-negotiate sharply.

Downtime of services, such as the carwash being broken or a gas station being out of gas, not only lead to missed business opportunities, but also damage to the company’s strong brand-names. The following examples are typical for what is met in every day business, hundreds of times every month in more than 23,000 gas stations across the US.

**Example** A carwash installation at a gas station was not functioning and in need of service for more than a week before maintenance staff was notified of this breakdown. During that week, the carwash did not contribute to company revenues.

**Example** A maintenance employee visited a gas station location to do routine checks on certain installations. The next week, one of the gasoline pumps breaks down. If the maintenance employee would have had insight in the power-use of the pump and had been be able to see that the power-use had been increasing considerably over the past few weeks\footnote{an increase in power-use is a sign for a pump which is about to break down}, the pump could have been replaced during a regular visit as a precaution. Now an extra, unscheduled visit, at extra costs, is needed, in addition to the loss of revenue since the system is non-operational for a period of time.

This lack of insight in critical operations has substantial financial consequences for the company which has over 23,000 gas stations in large parts of the United States. Sun’s iForce Ready Center was asked to contribute to a state of the art, technical solution which would unlock various sorts of information available within devices all over the company’s distributed business environment. This information is to be used by different actors involved, to optimize processes with the ultimate goal of lowering operational costs.

### 1.1.4 Problem summary

Throughout the entire E-Gas Station business environment, information available in devices doesn’t reach the right actor at the right time. Because actors do not have the right information at
the right time to steer and optimize processes, money is lost due to inefficiency every day.

1.2 Conceptualization of an architecture

This chapter describes the conceptualization of an architecture based on the case description presented in the previous section. The first section briefly presents a traditional solution and argues that it is not feasible for the E-Gas case. On following, the CarwashExample will be introduced, which is used to conceptualize the architecture, found in the final section.

1.2.1 A traditional solution: ERP

Based on the case previously described, many IT-consultants today, would recommend a centralized Enterprise Resource Planning (ERP)-system for storing the business state and business rules, in conjunction with a large data-warehouse for storing all the business events, to be used for trend-analysis and derivation of (longitudinal) management information. Centralized solutions as large enterprise ERP-systems have been implemented for almost a decade now and day to day practice learns that their centralized concept leads to system *inflexibility*. Apart from that, the concept of storing all information in a single data repository, in practice, makes retrieving the right information for a specific actor often like searching for a *needle in a haystack*. Finally, it may be noted that mining the data in a centralized repository is a very time-consuming process.

In the E-Gas case, implementing an ERP-system, together with a data-warehouse, will only solve part of the company’s problems. An ERP-system cannot meet the company’s demand for flexibility, multi-actor support and its geographical requirements, without bending the centralized concept of enterprise computing to inflexible, specifically configured connections between all kinds of different sub-systems. ERP-systems, as a centralized solution, do not do just to the geographic dispersal of 23,000 different gas stations, nor to the wide range of actors and their constantly changing information-need.

As an ERP-system does not meet all the requirements for a solution for E-Gas Station and proves to be an sub-optimal solution, this thesis explores new concepts, which better meet the requirements of organizations with many distributed business locations and need for a flexible solution.

1.2.2 The carwash example

To be able to conceptualize an architecture which meets the challenges of the EGas case, one particular example from the E-Gas station case is used throughout this thesis. Let us recall the previous example, depicted in Illustration 1.2.1.
To ensure operability of the car-wash at the gas station, the level of various fluids (soap level, wax level, etc.) in the machine have to be within specs. In this example, the clerk working behind the counter of the gas station needs to be informed when the soap level drops below a certain level. If so, he/she needs to refill the soap from the soap stock which is available in the gas station. Whenever the clerk uses some of the soap from the soap stock to refill the carwash, the amount of soap in the soap stock diminishes. Whenever the soap stock falls below a certain level, the procurement department at the oil company's headquarters needs to be notified, so they can re-order soap and to have the the stock refilled with the next scheduled delivery of supplies.

The various devices on the bottom of the picture (carwash and gas station) have properties which are relevant to the particular actors seen at the top of the figure. The main question is how the information available in devices is going to reach an actor present at an entirely different location.

The next section conceptualizes an architecture based on the example as just presented and the requirements as set by E-Gas Station management.

1.2.3 Conceptualization of the architecture

This section conceptualizes an architecture based on the example of the carwash installation as presented in the previous section. Over the following subsections, the concepts are added to the solution, until in the end the entire architecture has been presented.

Actors have a need for information from devices

As a starting point, it is recognized that actors in the business environment have a need for information available in devices at various locations to support their work, to allow them to control processes and make decisions. Illustration 1.2.2 depicts actors with an information-need and devices containing information, separated by a huge white gap. Currently information cannot not reach the actors in any way.
Devices have a virtual representation

In Illustration 1.2.3 the concept of the virtual device is introduced. Hiding technical complexity and variety, this concept introduces a common interface for access to information available in devices and wrapped to a virtual representation. Still, information does not reach actors.
Actors have a virtual representation

The virtual actor represents the physical actor in the architecture. Events which are received when the physical actor is off-line are stored, as well as granted rights and actor-specific preferences. Illustration 1.2.4 shows the virtual representations of various actors which allows them to be a part of the network-interaction. Now that actors and virtual devices are both brought into the architecture, let us look at ways to extract information.

Information-extracting filters deliver relevant events to virtual actors

A filter is a component which extracts information from one or more virtual devices in the framework and sends the information to the subscribed actor. Figure Illustration 1.2.5 shows that special listening filters listen to the virtual gas station and virtual carwash and send business events to certain actors.
Access to system functionality is restricted by an administrator

In a distributed environment stretching many organizations, it is of importance that access to information and system functionality is restricted to authorized users. Illustration 1.2.6 introduces the virtual administrator, who is the gateway to the physical administrator. The physical administrator is the one who grants access to system functionality or information in his organizational domain, relevant both for systems administration, as for business administration.
An actor interacts with system functionality through a business rule

A business rule defines the specific system functionality for one or more virtual devices, which allows an actor to act on the system. Illustration 1.2.7 shows the maintenance employee on certain carwash functionality, through sending a business action to a business rule. The maintenance employee might remotely reset the device.

Various GUI-panels allow actors to receive events anytime, anywhere

To support reception of information by actors when they are on the road, or at a different location, the concept of the GUI-panel allows actors to interact with their virtual representation through various means. The GUI-panels depicted in Illustration 1.2.8 could represent a workstation, mobile phone, internet-kiosk or perhaps a pager.
Chapter 1       Introduction of the basic architecture

An architecture as solution

As a summary to the introduction of the architecture, numbered items in the following list reflect the various concepts introduced to a solution for the E-Gas case. Illustration 1.2.9 on page 13 depicts a graphical representation of the numbered items presented.

- **Information generating (real or simulated) devices are wrapped to a virtual representation called a virtual device, making their relevant properties available within the architecture.**
  Device-specific implementation is hidden from the architecture, abstracting from technical complexity and creating a single interface for addressing information throughout various organizations. As such, virtual devices could be connected to real, legacy devices, simulated devices, or any information source.

- **Actors interact with the system through their virtual representation called virtual actor.** The virtual actor represents the physical actor in the architecture, so events which are received when the physical actor is off-line can be stored. Apart from that, the virtual actor stores granted rights and received business events and is able to process business events based on additional preferences set by the virtual actor.

- **Actors are notified of events through filters, to which they can subscribe.** Introduction of special listening filters not only allow actors to query a device for a specific property-value, but also support actor-notification of relevant system-events, discarding all irrelevant information.
• Access to information available in virtual devices or particular system functionality is restricted by an administrator, which also has a virtual representation. The concept of information-ownership and an information-gatekeeper who guards access to information or functionality available within his/her organizational domain allows organizations to share particular information while hiding other information.

• An Actor can control\(^\text{5}\) the system (one or more virtual devices) by sending a business action to a business rule. The introduction of the business rule allows specific functionality to be brought into the architecture for single virtual devices, as well as for multiple virtual devices in a distributed environment.

• Actors are able to receive events while they are on the move by the concept the GUI-panel. A GUI-panel might be a workstation, but also pager or mobile phone, equipped some sort of wireless access protocol. A loosely connected GUI-panel disconnects the virtual actor-state from the way information is presented.

Virtual representations of the various information generating devices found throughout the organization are brought into the architecture where information extracting filters listen to these properties. As soon as a property-change is relevant to one or more actors, a business event is sent out to the actor’s virtual representation. Based on the business event the actor can act or

\(^{5}\) Invoke an action which causes a state change.
may decide that no immediate action is eminent.

To stay informed of the conditions that require their action in the carwash example, as mentioned in section §1.2.2 on page 6, respective actors subscribe to a particular \textit{ThresholdFilter} which listens to the relevant property and sends a message as soon as the value drops below the indicated threshold. A graphical overview of the example emphasizing the various types of filters and components is found in figure Illustration 1.2.10. The figure also illustrates the architecture's concepts are equipped to extract information from both real, as simulated carwashes.

The E-Gas case and the carwash example just mentioned, is used throughout the entire thesis to translate the concepts that are introduced to case specific-examples, which have been implemented in a working demo.

The next chapter will now introduce the research that has been performed on extending the basic architecture with visualization and simulation.
2
Introduction of the research

The previous chapter presented the EGas case, which was the starting point for a basic architecture which objective could be described as *improvement of the usability of the information produced by a network of (electronic) devices*. Furthermore, it was identified that this challenge originated from different objectives and actors. For the iForce Ready Centre itself it was identified that the challenge originated from the objective to increase the *ability to demonstrate* Sun’s solutions. For Sun’s customers (in particular Egas, see previous chapter) it was identified that the challenge emanated from the objective to decrease costs, by increasing the use of device information as *support for decision-making*. However, despite these different perspectives, some common basic requirements could be identified, like the need to connect an underlying network of information producing devices. It is for these common requirements that a generic architecture has been developed to meet these stated objectives. This architecture was introduced in the previous chapter.

Although the basic architecture is able to make information accessible in a network of devices, it does so in a relatively primitive way, by displaying event content as simple text. The research presented in this report, was focussed to define some extensions on top of the basic architecture, in order to increase its usability for actors that need to use the information it produces for their business decisions. The next section introduces the architecture extensions. After that, the development methodology used during the research is presented. On following, the research question is introduced. Finally, the report structure is introduced.

2.1 The architecture extensions introduced

This section introduces the architecture extensions presented in this report. Since these extensions are pointed at providing increased support for decisionmaking, the next two sections introduce the concepts of decisionmaking as used during the research. After that the extensions that will be presented by this report are introduced. Finally, some first consequences of these extensions for the architecture are presented.

---

6 A device in this sense can be any electronic device, from coffee-machine to airplane, as long as it produces information and is in some way connected to a network.

7 Since this architecture has a lot to do with the exchange of information, the architecture has been named the *Narad architecture*, after the Indian god of messaging.
2.1.1 Different contexts, different decisionmaking

It has been presented in the introduction that the basic architecture should enable the oil company to optimize its processes in order to reduce costs. It has also been shown that in such a company, the responsibilities for these processes are distributed among many actors and organization layers. It may be noted that the type of decisions made by these actors is related to the processes they are involved in. This is also reflected in [SOL00], which presents a classification of different organizational focuses for decision-making and system design. This classification is reproduced in the table below:

<table>
<thead>
<tr>
<th>Level ↓</th>
<th>Managerial interest →</th>
<th>Control</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro (workplace level)</td>
<td>Improving primary tasks</td>
<td>Improving control tasks affecting execution of tasks at the workplace</td>
<td>Developing a vision on the purpose of tasks and (sub)processes within the business system</td>
</tr>
<tr>
<td>Meso (department level)</td>
<td>Improving primary processes within the organization</td>
<td>Improving control processes, e.g. coordination within the organization</td>
<td>Developing a vision on the purpose of the business (sub)systems within its ‘own’ context</td>
</tr>
<tr>
<td>Macro (organizational level)</td>
<td>Improving primary tasks and processes between organizations</td>
<td>Improving control processes between organizations, e.g. interorganizational coordination</td>
<td>Developing a vision on the purpose of the business system in its environment</td>
</tr>
</tbody>
</table>

*Table 1: Different design’s focuses*

It may be noted that along the dimension of managerial interest and organization level, the variety in the system of interest and the number of actors involved in the decisionmaking process will rise. For example, a gasstation clerk responsible for the availability of a carwash, only needs information on the functioning of the carwash and decides to notify an actor responsible for the maintenance process if the carwash is not functioning. To make this decision, his information need is bounded and no other actors need to be consulted to take this decision. It is therefore possible to automate such a decision, by installing an electronic listener that will send an e-mail to the actor responsible for maintenance when a failure occurs.

---

8 A design focus is regarded to be a decision focus as well.

9 This example leaves out the other processes the clerk is responsible for and the complexity involved in actually repairing a non-available carwash.
The CEO of the company, however, decides on the oil company’s strategy: what capital goods should the company invest in, what business coalitions should be formed, what are the purposes of the business system in its environment. It may be clear that the variety of the situation to be considered is much higher than in the previous example. Also, multiple actors, having different interests, are involved in these decisions. It is highly unlikely that these decisions may successfully be automated, following Ashby’s law of requisite variety (see [BEER95]), which states that the varieties in the system being controlled, can only be controlled if the control signals match this variety. The level of variety in the system of interest for the CEO (e.g. the oil company in its environment) is much too high to automate his decisions. However, a system that provides the CEO with aggregated information on the company’s results, may still be helpful for the CEO needing to decide on the future of the key activities of the company.

It is this modest approach towards supporting the actor’s information needs during decisionmaking, that has been the starting point for the architecture extensions introduced in this report. The focus has been to identify those aspects of the decisionmaking processes, that might actually be supported by the functionality offered by the basic architecture: providing specific information regarding devices at the right time to the right actor. The notions on decisionmaking introduced in the next section has been used to accomplish this.

**Notions on decisionmaking**

In order to identify these aspects, the theoretical model of the problem solving (e.g. purposeful decision-making) process presented in Illustration 2.1.1 has been used as a guideline. The model is based on [BOTS97] and [SOL00]. This model breaks the problem solving process down in several *activities* (the dark ovals) and *outputs* (the rectangles). The
model seems to imply a strictly cyclical process. However, the two requirements connecting the outputs 'conceptual model' & 'solution' and 'problem perception' & 'empirical model' function as checks for eventual process iterations. The model also assumes that the implementation of a solution never completely solves all problems. This is expressed in the model by the fact the end of the 'evaluation' activity may result in a new problem perception.

In addition, Illustration 2.1.2 illustrates the need of an actor to filter information, when making decisions. Only a part of the information that’s observable, is relevant to decide for a given problem. To recall the example of the gasstation clerk: he does not need to be notified of the water used by the carwash, he only needs to check that the carwash is functioning. It may be noted that this aspect of decisionmaking is already implemented in the filter component of the basic architecture.

Finally, the five notions on necessary conditions for effective control have been noted, as formulated in [LEEUW90]:

- Clear objectives of the control process.
- Availability of a model of the controlled system.
- Available information on the state and environment of the controlled system.
- Availability of enough steering instruments.
- Availability of sufficient information and communication handling capacity.

2.1.2 **The architecture extensions identified for support of decision-making**

Now the theoretical background and a first specification of the decision types have been presented, it has become possible to introduce the two extensions that have been identified for the architecture, namely support for *visualization* and support for *prediction*. 
Visualization

One of the notions of necessary conditions for effective control is the 'Availability of sufficient information and communication handling capacity'. This capacity, however, is influenced by the way information is presented. According to [GORDON89] and [VREEDE96], visualization and animation of information significantly increases the effectiveness of communication of information. Recalling the fact that the potential information streams in EGas are potentially voluminous, visualization has been considered in this report as a useful extension of the basic architecture in order to improve its usability for decisionmaking. The architecture extension that should provide the visualization capabilities is the Visualization & Control system (V&C system), which will be presented in chapter 3. It may be noted that 'control' has been included as the logical counterpart of visualization: while a visualization component enables an actor to retrieve information from a device, the control component enables that actor to forward (command) information to the device. Addition of the control functionality also makes sense given the control condition of 'Availability of enough steering instruments'. The location of the V&C system extension of the architecture is shown in Illustration 2.1.3.

Prediction

Besides visualization, a need for support for prediction has been identified, given the actor's need to evaluate alternative courses of action, which may not always be easily evaluated. An example from EGas is the gasoline distribution between storage points and local gasstations. Although the distribution process manager may feel that alternative routing algorithms may improve performance, he will be reluctant to implement it, given the huge business interest of an uninterrupted gasoline supply for the oil company.

The concept of simulation has been identified as a potential framework extension to provide the functionality of safe prediction. Simulation has been considered to be well suited to extend the basic architecture, given the fact that the virtual device layer (in fact already a virtual presentation or model) in the architecture may provide a useful starting point to generate an empirical simulation model. The initial location of simulation as an architecture extension is shown in Illustration 2.1.4. It may be noted that the V&C system, as
introduced in the previous section, provides the animation of the simulation model. The research performed on simulation as an architecture extension is presented in chapter 4.

**Wrapper component**

A final extension researched in this report, is the wrapper component, which provides the communication between simulation or real device and the architecture. Since this component is highly device specific, it is developed as a 'plug-in' component for the Virtual Device. The location of the wrapper component in the architecture is illustrated in Illustration 2.1.5. The research performed for the wrapper component is presented in chapter 5.

2.1.3 The framework extension in the decisionmaking model

The model presented in Illustration 2.1.6 is an extension of the model shown in Illustration 2.1.1. It adds a division of the decisionmaking process into four distinct phases: evaluation, specification, design and implementation. Each phases consists of activities and outputs. The requirements shown in Illustration 2.1.1 are left out for clarity. Also, some outputs are 'shared' between phases, since they are crucial output for one phase, as well as crucial input for the onfollowing one. The reason for introducing the decision-making phases is twofold: it makes the decision-making model more manageable and it adds new information, namely the nature of the underlying system being controlled by the V&C system during the different phases of the decision-making process. During the evaluation and specification phases, the actor will use the V&C system to extract information from the real business system. After the specification of the empirical situation, however, a simulated model of the relevant real system should be generated or developed. This model may then be used to generate solutions and to predict the performance of these solutions. This will be discussed in more detail in the next chapter. What is relevant for now, however, is the fact that the V&C system should be usable for the real as well as the simulated system, since the V&C system is not concerned with the nature of the underlying devices. This idea is illustrated in Illustration 2.1.7. In the situation depicted on the left side of the picture, the actor has instantiated a view on the system that enables him to visualize the operations of two real gaspumps. Let us for example say that his motive for doing so, was the observation of a very high pump utilization for a substantial period of time and that he would like to know whether the purchase of another
gas pump would be profitable. By selecting the two real gas pumps and by instantiating a filter that measures the pump utilization over the day, the actor has completed the specification phase for this particular problem description. The next step in the decision-making process is the generation of solutions. In order to do so, the actor might generate or develop a simulation model of the two real devices and the customer arrival pattern. Such a model would enable him to add another simulated gas pump and predict the results. Now, in order to visualize the simulated gas-
pumps, only some *copies* of the same basic components used during the specification phase are needed. This is possible, because the V&C system operates on representations (e.g. virtual devices) of devices in the underlying system. These representations (e.g. virtual devices) do not necessarily change when the underlying real devices are replaced by simulated ones. After all, they will still expose the same properties and methods. This characteristic of the V&C system may greatly reduce the costs of decision-making, since no new visualization & control system needs to be developed for the simulation system. It may also be noted that a combination of real and simulated devices can be controlled with a single V&C system. This characteristic may be used during the implementation phase, which is discussed in §4.1.1 on page 89.

### 2.2 Design methodology

This section reflects on the design process during the project using a way of framework ([SOL00]), describing the way of thinking, way of working, way of modelling and way of controlling, as depicted in Illustration 2.2.1.

![Illustration 2.2.1: Way-of-framework](image)

#### 2.2.1 Way of thinking

The way of thinking for the project is best characterized by the theoretical notions found in the previous section. In short, the following notions are illustrative for the way of thinking:
• Multi-actor view on various parties involved in business.
• Actor information-need based on problem-perception.
• Dynamics of organizations, business environments.
• Business processes transcend single organizations.
• A systems perspective on organizations and (inter-) organizational processes.
• The system notifies the actor, instead of the actor constantly queries the system.
• Open standards and well-defined interfaces, wrapping of components allowing legacy devices to be a part present of the solution, instead of being the problem.
• Actors are responsible, so actors (not the system) should be empowered to make decisions.

2.2.2 Way of working

The way of working during the research project is characterized as a generic middle-out approach. This section describes three different approaches the E-Gas business case; top-down, middle-out and bottom-up. It argues why the middle-out approach was most applicable for this case. Different ways of approaching the E-Gas business case are:

• Top-down: A top-down approach would start with interviewing various actors in the business chain, examine high-level (maintenance) procedures, inter-organizational contracts, various suppliers in different area’s, etc.

• Middle-out: The middle-out approach starts at the intermediate-level of one gas station, identify relevant devices there and match these with actor information-need and business processes.

• Bottom-up: The bottom-up approach would start looking from individual devices, their specific interfaces and information exchange, from there look up to match the information with specific actors.

The middle-out approach, focusing on the gas station level was considered most applicable for the following reasons:

• A lot of the cost-intensive, inefficient, support and maintenance processes are related to operations at gas station level

• A gas station, at the end of the business chain, is where the money is made by selling products and services to the customer

2.2.3 Way of modelling

As ‘Way of modelling’, UML-techniques (see [FOWLER00]) for modelling component types and
interactions between them have been used.

2.2.4 **Way of controlling**

During the project, at several delivery moments, progress was established by freezing the conceptual idea’s during a joint session with either the coach at Sun, or the coach from Delft University of Technology. Based on these sessions, direction and targets for the following weeks were discussed and established.

2.2.5 **Way of working**

The way of working is characterized by a combination of an iterative and an iterative approach. The incremental process is characterized by the following steps:

1) Analysis of the actor’s use of the architecture
2) Identification of functional and technical requirements
3) Conceptualization of the solution and its implementation in the Java-language
4) Reflection on the solution and its implementation, resulting in design choices

Step 3 in the incremental process was iterative in nature

2.3 **Research question**

The initial research question of this report, based on the scope just described is the following:

*How should a basic architecture, that makes raw information in a network of distributed devices accessible, be extended in order to meet the actor’s objectives of the architecture, in particular demonstration and decision-making?*

This directly poses the question **what kind of** extensions are needed in order to meet the objectives of the architecture. This question already has been answered in section 2.1.

The first extension identified is a **visualization & control** system, implied by the requirements of visualization, interaction with and monitoring of the system. This system should somehow be linked to the underlying network of information producing devices, given the requirement for coupled visualization and system interaction.

The second extension identified is **simulation**, implied by the requirement to predict system behavior, for several scenario’s. Initially, simulation will be defined (following [ATOMICA01]) as an:

*Imitation or representation, as of a potential situation or in experimental testing.*

Since these extensions are potentially able to meet the requirements identified in , we can regard
the question of the *nature* of the architecture extensions to be answered. Therefore, the final research question turns out to be:

*How should the basic architecture, that makes raw information in a network of distributed devices accessible, be extended with a visualization & control system and simulation capabilities in order to meet its objectives of demonstration and decision-making?*

This final research question leaves room for a variety of concepts and implementations of both the visualization & control system as well as simulation. It has indeed been the goal of this research to both map part of this variety as well as recommend some choices available. Another goal has been to answer the ‘how’ question practically, not just theoretically. Therefore, a so-called lab implementation has been built for each subsystem.

The final paragraph of this chapter will introduce the general structure of this report and the way the research question and the research goals have been applied to the proposed architecture extensions.

## 2.4 Structure of the report

As illustrated in the previous section, the architecture extensions are central to this report. The report’s chapter division is therefore governed by the architecture extensions. Apart from the architecture extensions mentioned in the research question (e.g. visualization & control system and simulation), the *wrapper* subsystem is discussed as well. Although the wrapper subsystem does not allow direct actor interaction itself, it is an indispensable subsystem for presenting actual representations of the underlying real and simulated devices to the actor.

The research question will be answered *separately* for the three mentioned subsystems and *integral* in the concluding chapter. This structure, including the relation with the general introduction, is shown in Illustration 2.4.1 below:
Finally, the way the research question will be treated per subsystem is explained below:

0. Introduction The introduction presents a short overview of the features of the developed subsystem. It also describes the relation between the subsystem and the rest of the architecture.

1. Design requirements

The section on design requirements will discuss the following themes:

(a) Exploration of user requirements and applications.

In this section the general user requirements will be broken down in subsystem specific user-requirements. The consequences of these identified user requirements for the subsystem are explored, this in combination with relevant notions from theory and cases. Since this research has not focused on the identification of user requirements, not all of them
may have been identified. However, the ones most relevant from an implementation oriented point of view should be included.

(b) **Distillation of technical requirements**

Based on the identified user requirements, technical requirements are identified that should be met by the subsystem.

(c) **Design choices:** In order to meet certain requirements, multiple implementations may be possible. This section discusses some of the design choices encountered.

2. **Lab implementation**

Of each subsystem, a lab implementation has been built (and included on <ref CDROM>). The challenge of the lab implementation was to explore the technologies available to determine to what extent the requirements could be met. Therefore, this section will focus on the technologies used and implementational issues encountered. If the implementation has been used for one of iForce’s cases, this will be pointed out.

3. **Conclusions & recommendations for future research**

Based on the experience with the lab implementation and the requirements identified, the basic research question will be addressed. A range of conclusions and recommendations can be expected here, among which the following:

- Some requirements are impossible to meet (not yet encountered)

- Some requirements have not yet been met, but could be realized after more research. Technologies may already be available.

- Some requirements have already been partly realized in the lab implementation. For these requirements technologies and methods to implement them are available on <CD>, appendices or other sources. Further research may however be required, either to improve the performance of the lab implementation or to develop new technology to ease implementation issues.

- Some requirements have been fully realized in the lab implementation. Technologies and methods to implement them are available on <CD>, appendices or other sources.
This chapter presents the research conducted for the development of the visualization & control system (V&C system). Illustration 3.1 shows the relation between the V&C system and the basic architecture. The system has indeed been developed as an extension on the basic architecture for distributed devices as introduced in [HISSINKMULLER01]. This basic architecture makes information accessible in a network of information producing devices. It may be recalled that the architecture does so by making available ‘virtual devices’ that represent underlying real or simulated devices (see Illustration 1.1.1). This implicates, as Illustration 3.1 shows, that the V&C system does not have to be aware whether the underlying devices are real or simulated.

The V&C system should add interaction with the actor on top of the basic architecture to meet his objectives. Visualization contributes to this, since it eases the interpretation of the information presented by the architecture (see Intermezzo 1: Visualization concepts on page 29). As discussed in the Introduction of this report, the identified objectives of the actor are:

- Increased support for demonstration (of technical aspects).
• Increased support for decision-making.

The sequence in which these objectives are states is not incidental, but coincides with the chronological course of the development project. The first visualization results were achieved with the objective of technical demonstration in mind and focused on direct visualization of device properties. This is best illustrated by the 'Current States' subpanel in Illustration 3.1. This mode is also fully supported in the lab implementation. However, during the project the concepts on visualization and control evolved in order to be able to also support the second objective: support for decision-making. It was during this evolution, that the theoretical foundations on systems engineering, decision making support and business engineering\(^\text{10}\) contributed to the full scope of the visualization and control system presented in Illustration 3.1. Apart from visualizing current states, that system also provides the actor with the ability to build and use prediction models, to interact with the system using dynamically extracted control components (the buttons in Illustration 3.1) and finally to visualize the context of the system of interest\(^\text{11}\). This additional functionality presented is not yet realized in the lab implementation. However, the basic implementation can serve as a starting point for further development.

The functionalities offered by the system will be further discussed in the next section, that focuses on the identification of design requirements and choices. Section 3.2 introduces the lab implementation that has been realized during the project. Finally, the concluding section addresses the research question and recommends directions for further research and development of the V&C system.

**Intermezzo 1: Visualization Concepts**

The Atomica dictionary defines 'to visualize [something]' as: 'to form a mental image of [something]'. This definition implies the notion that people do understand something by forming a mental image, or internal model of what they observe. In this sense, the function of a visualization system is to support this process of mental image forming, by presenting information\(^\text{12}\) in a way that is easily 'translated' into a mental model. Or, in other words, the function of the visualization is to communicate information with its user (a human actor) effectively. From this it follows that the concepts used to visualize information, should comply with the way humans perceive visual information presented to them. Research on this subject is for example described in [GORDON89].

---

10 As introduced in [HISSINKMULLER01].

11 This corresponds with the structural overview panel in Illustration 3.1

12 It may be noted that information presentation does not necessarily have to be visual: audible information may also support the forming of a mental image
The discussion thus far assumes a static information presentation. However, information may also be presented dynamically, in which information is constantly updated with changes in an actor’s preferences or information sources. In [SPENCE01] a model (reproduced in Illustration 3.2) is introduced that describes three functions that should be performed by a system that dynamically presents information to an actor. The identified functions are selection, encoding and presentation.

**Selection** has to do with the way the visualization system should filter and collect information. The collection part of the selection defines what data sources should be used. The filter part of the selection forms the selection criteria for the visualization system. A useful set of criteria will pass information that’s relevant for the actor and discard all other information. The field of systems engineering, as introduced in [HISSINKMULLER01], has identified several useful ways to define these criteria, notably definition by sub-, aspect- or phasesystem.

**Encoding** has to do with the way data is to be transformed into visual information presented to an actor. [SPENCE01] presents several encoding schemes and their corresponding visualization components. He illustrates that data may be encoded in multiple ways and with different visualization objectives in mind. The encoding should be chosen in such a way that the visualization emphasizes the most relevant aspects for the actor. Some examples may illustrate this:

- If the visualization objective is to reduce data complexity, a boxplot diagram may be used. In this way a data-set of a large number of points is reduced to a bounded average.
- If the visualization objective is to illustrate (historical or predicted) trends, a graph may be the most suitable visualization.
- If the visualization objective is to visualize relationships between several dimensions, a combination of 3D graphs, color and sound encoding may be used. Pie-charts or mosaic displays may be used to encode the ratio or significance of multiple values.

![Illustration 3.2: Functions of a dynamic visualization system](image-url)
If the visualization objective is to visualize the performance of a system, information may be encoded with colors or faces. Both colors and faces have a normative connotation. The color ‘green’, for example, is usually associated with ‘good’ performance, as opposed to the color ‘red’. With faces, a ‘happy’ face (☺) is associated with ‘good’, a ‘sad’ face (☹) with ‘bad’ performance.

If the visualization objective is to offer a realistic representation of a system, a visualization using Virtual Reality may be appropriate. If the system of interest already exists, an image or movie of the actual system can also be used. This type of visualization is often used to provide the actor a spatial and structural understanding of the visualized system. This may serve as the context for other information presented to the actor.

The presentation part, finally, deals with the way a large collection of information items should be presented to an actor. The actor should also be able to navigate through the collection and be able to see the structure of the relationships between the elements. [SPENCE01] introduces several concepts and ideas to realize these presentation aspects:

- Focus vs. context. The visualization system should detail the information presented in a focused part of the presentation, while lessening detail of the information available in the context of the focused part. An example of a tool providing this functionality is a magnification tool.

- Zoom & panning. A large collection of information items may contain different relationships on different aggregation levels. A tool that allows an actor to zoom in or out will make these relations available for the actor.

- Network layouts. A system may have the characteristics of a network. This occurs when the individual elements are connected to other elements. Common examples are a computer network or a railway infrastructure. Networks may also be more abstract, for example in the case of a semantic network of concepts. The presentation of a network raises the challenge of visualizing the network with the minimum amount of cross-overs and occlusions. An example of a modern approach towards layout and navigation in abstract networks is the hyperbolic browser (see [INXIGHT01]).

Although the visualization functions may be performed automatically by the visualization system, the actor should be able to alter the strategy used to perform them. This is illustrated in Illustration 3.2 by the set of parameters the actor passes to the system. This parameter passing does not have the nature of a one-time initialization, but should rather be described as a continuous inter-
action between the visualization system and the actor. Illustration 3.3 presents the model (based on [SPENCE01]) that describes this interaction as a cycle of activities and outputs. The actor is supposed to perform the tasks of modeling, interpreting and strategy formulation. The modeling activity is the process in which the actor tries to ‘absorb’ the delivered content into his\(^\text{13}\) internal model. The content may be a trigger for the actor to re-interpret his internal model, which then results in a new interpretation. That may lead to a new need for information, resulting in a new browsing strategy (compare with the parameters in Illustration 3.1). This will trigger the visualization system to produce new content (the ‘browse’ activity is performed by the visualization system and is equal to the process described in Illustration 3.1).

The conclusion of this intermezzo therefore is that a visualization system should be able to perform the functions of selection, encoding & presentation in a process of continuous interaction with the actor.

### 3.1 Design requirements

This section explores the design requirements and choices for the visualization and control system. It starts with two paragraphs that focus on the functional requirements posed by the actor on the V&C system, in line with actor’s objectives of the architecture, namely demonstration and support for decision-making. Section 3.1.3 discusses some technical requirements that can be derived from these functional requirements. Finally, 3.1.4 discussed some design choices that have been encountered during the project.

#### 3.1.1 Actor requirements for demonstration

Sun’s iForce center often demonstrates the workings of a reference architecture system to their customers. Usually the demonstration should prove that technical requirements like throughput, or peak load are met by the system. However, it is often hard to communicate the implications of the output generated by such a system. Common output consists of log-files, textual output or

---

\(^\text{13}\) The phrase ‘his’ in this report is usually meant as a short-hand for the phrase ‘his or her’. 
even just a blinking light on a server, indicating some sort of activity. These kinds of output do not enable the customer to form an internal conceptual model of what’s going on inside the system (compare Intermezzo 1: Visualization concepts on page 29). The remainder of this section will discuss the requirements for the V&C system that are implied by this use case. These requirements have largely been identified during the work on the demonstration of Sun’s Data Management Center (DMC), described in [JACOBS01]. The background and results of this case can be found in Appendix 6.1.4. The requirements identified are: a selection mechanism, coupled visualization, interaction, meaningful representation and dynamic panel generation. They are illustrated in the following sections by the example implementation shown in Illustration 3.1.2.

A need for selection

A typical customer configuration usually consists of several components, see for example the sample configuration in Illustration 3.1.1. However, not all the system components are crucial for the total system performance. It is for example quite plausible that the key factor determining the performance of the sample configuration is the database cluster. The demonstrating actor may therefore need to visualize only some properties of one of the database servers. The next paragraph will illustrate the consequences of this requirement for the Narad architecture.

The Narad architecture basically assumes that each device on the network is made available by use of a virtual device (see chapter 5, Wrapping devices on page 114). This means that a Narad implementation for the system shown in Illustration 3.1.1 would already consist of about twenty virtual devices. The demonstrating actor should therefore be able to notify the Narad architecture that he’s only interested in selected devices (for example the database server) and on top of that,
in selected properties (for example system load) of those devices. In other words, the actor should be able to construct a topic of interest. Such a topic should enable the actor to define a subsystem of interest (e.g. a subset of devices) and to define an aspectsystem of interest (e.g. a subset of aspects or properties). The sample implementation shown in Illustration 3.1.2 therefore consists of a topic object that is located inside the visualization panel. The given topic instructs the panel to process just the events that match with its topic. In this way the visualization panel will process only a subset of the events arriving at the virtual actor component. It is shown that actually more events are propagated to the virtual actor component, since the filter propagates events from both virtual devices. The filter component is also able to perform additional filter operations, see [HISSINKMULLER02] for detailed information.

In this example, the demonstrating actor probably knows quite exact how his topic of interest should look like. However, this is not the case in general. If the actor does not know the exact form of the topic of his interest, the visualization system should offer the possibility to browse for information (see Intermezzo 1: Visualization concepts). As presented in [JACOBS01], the Narad architecture supports the browsing of components registered within the architecture. However, if the Narad architecture is deployed in a large system (for example: including all US gastations in the architecture would results in around 20.000 gaspumps!), simple browsing may result in a significant amount of requests for the underlying virtual devices. In order to prevent this, the devices could be organized in domains. In this way the architecture could restrict the actors to browse all 20.000 devices continuously. These domains could be organized in several ways, for example by geographical location or by organization unit. Since these domains often overlap (a device is both located in a geographical area as well part of a certain organizational unit) this implies that (virtual) devices should be allowed to belong to multiple domains at the same time. It should also be clear which and whose domain definition is referred to by a device’s domain description. After all, many parallel domain definitions exist, for example because each organization administers its own organizational domain structure.
Coupled, real-time visualization

The demonstrations of the iForce center often have to prove that certain key performance indicators (like maximum number of concurrent users) are met by the system. Usually the customer has seen a lot of specifications, both from Sun and its competitors. However, these specifications are not always convincing for the customer, due to lack of information on test procedures and measurements. The essential characteristic of iForce’s approach is therefore the demonstration of actual, up and running configurations. To support this kind of demonstration, the visualization therefore needs to be directly fed by the demonstrated system. Also, the update speed of the visualization components should match the speed of changes in the demonstrated system. Since a typical computer system may show highly dynamic behavior (for example with regard to available memory), this implies real-time (or at least as fast as possible) updating. The main part of this functionality is realized by the basic architecture, which enables actors to subscribe to events that are delivered by the architecture as soon as they arise.

Interaction

A customer demonstration usually involves starting some scenario that creates some load on the demonstrated system. In case of demonstrating a database server, the scenario may consist of a sequence of database queries. The V&C system could provide a control element that would allow the demonstrating actor to formulate and execute the queries during the demonstration, in dialogue with the customer. In this way, the customer’s involvement with the demonstration may be increased. The possibility of a control element is shown in Illustration 3.1.2 by the ‘action’ button in the ‘current states’ panel. The actual communication with the underlying real device is realized by means of a business rule component. Such a component is allowed to perform some kind of action on the underlying virtual and consequently real device. The business rule component is treated in [HISSINKMULLER02].

Meaningful representation

The main function of visualization during a demonstration is to improve the ability to communicate information about the system being demonstrated. Intermezzo 1: Visualization concepts mentions that the way information is encoded should match the visualization objective for that type of information. The different types of information communicated during a demonstration therefore imply that a visualization panel should consist of different types of visualization components, that encode information in specific ways. This is illustrated in Illustration 3.1.2 by the presence of two types of visualization components, namely level indicators and state animation components. The level indicators communicate the current value of a property. Apart from the plain value, the
significance of the plain value and a normative evaluation is also presented. The significance of
the value is communicated by visualizing the value in relation with a representative upper and
lower bound. The normative information may be encoded in the color of the level-bar.

The state animation component communicates different information. The image of the server
presented to the actor serves as a representation of the actual device and therefore provides the user
with a context for the other information presented to him. The state animation component also
presents the current, discrete state of the device. For example, if the underlying device is busy,
the state animation component may show animated flashing lights.

It can be concluded that the information encoding in a visualization component should match the
visualization objective for that information.

**Dynamic panel generation**

Thus far it has been discussed that an actor should be able to define his topic of interest. Based
on this topic, events are propagated to his visualization panel. It has also been discussed that a
visualization panel contains different types of visualization. Furthermore, a visualization compo-
nent may also contain device specific information. The state animation component for example
contains an image of the underlying device, while the level indicator’s upper and lower bounds are
also specific for a certain property (for example, a negative value for processor load is not possible).
Little has been said this far, though, about how the visualization panel itself should be developed.
However, given the dynamic nature of the actor’s needs (supported by the topic concept)
and the close relation between visualization component and device it seems most natural to gen-
erate the visualization panel dynamically based on a topic provided by the actor. Also, in order to
support the device specific visualization characteristics, visualization components could be
administered within the architecture, close to the virtual devices. This requires that the architec-
ture offers functionality to store by and retrieve components from the architecture components.

On the other hand there is also a need for the actor to customize an automatically generated visu-
alization panel. First of all, an actor may want to alter normative boundaries of the visualization
components. In case of the level indicator this implies altering the limits governing the level color.
Besides this, an actor may also want to replace an auto-generated component altogether, simply
because it does not encode information in the right way. Therefore, the V&C system should also
support runtime customization.

### 3.1.2 Actor requirements for decision-making

The final sections of this paragraph identify requirements for the V&C system for each of the
decision-making phases illustrated in Illustration 2.1.7, by focussing on the way the V&C system
may support the identified activities and outputs. Finally, active control components are introduced, as a means to automate standard decisions.

**Evaluation phase**

During the evaluation phase, the actor examines the performance of the business system. He does so by applying a normative context to the information delivered by the system and by translating system information into a formal problem description that may serve as the input for the next decision-making phase: the specification. The normative context of an actor may consist of a set of objectives, that govern the actors need for information. The problem specification, however, is not a simple translation of objectives into information needs, it is rather an iterative process in which the availability of information and instruments also influences the objectives used by an actor to structure and rationalize the problem description. A separate environment may be provided to support the actor with the structuring of objectives and information needs. This section presents some of the user requirements for the V&C system for this decision-making phase.

**Management by exception**

An actor responsible for a certain system while seldom evaluate the behavior of all system parts or aspects. The system under his control may just be too large or complex to do so. Instead the actor evaluates system performance by applying a normative context for the system’s behavior. This normative context serves as a *classification* of system performance, which enables an actor to focus on that part of the system that actually requires his attention. An actor may classify the system performance along (at least) two dimensions: expectation and desire. Such a classification enables him to determine whether the measured system performance is as expected\(^{14}\), worse than expected, or perhaps even better than expected. The attention of the actor may only be required when the performance is worse or better than expected. In other words: the actor only wants to be notified of exceptions of ’normal’ system performance.

This requirement implies the following consequence for the architecture and the V&C system. First of all the architecture should enable the actor to put his normative context into effect. This is already implemented in the ’filter’ component. This component enables an actor to define complex filtering routines, that in effect determine whether incoming information from the devices should be passed on to the actor or not. This functionality is described in more detail in [HISS-INKMULLER02]. The second consequence is that an actor should be able to instantiate ’silent visualization’, illustrated in Illustration 3.1.3. This can be defined as visualization that is not shown when there is no situation requiring the actor’s attention, but is activated the moment the

---

\(^{14}\) Note that a system may behave as *expected*, but at the same time not as *desired*. 
actor’s attention is required. This can be regarded as an extension of the filter component. The filter may be designed by an actor to pass only information requiring the actor’s attention. As soon as such a filter fires an event, the accompanying visualization should be activated and initialized. This may require that a filter keeps track of history events, in order to be able to initialize all the visualization components that are part of the ‘silent visualization’. These events should be fired as the activation occurs.

**Structural overview**

During the conceptualization activity, the actor searches for a model that enables him to control the system (as described in [HissinkMuller01] and [Bots97]). Such a model should focus on the relationships between instruments, environment and criteria derived from the actor’s objectives. This implies that the mere visualization of an underperforming device does not provide the actor enough information to make decisions to solve that problem. Instead, the structure or context in which the device operates should be visualizable as well, to provide clues for environmental influences and possible solutions. Illustration 3.1.4 shows an example of such a structural overview of a system, in this case a gasstation. Let us assume that the gaspumps have high utilization (above 98%). This might be of interest to the responsible actor, since this may
Simulate & Distributed Visualization

39

imply that queues form for the pumps, which could be resolved by adding more pumps. In this case the mere visualization of the high utilization of a single gaspump would not help the actor much. A structural overview as presented in Illustration 3.1.4, however, provides him with more information: the spatial layout gives clues about the potential expansion of the gaspump park, which is a possible instrument for this actor.

Another situation requiring contextual information is that of the transfer of system knowledge. If an actor is delegating his responsibilities to another actor, for example because he retires, he needs to transfer his knowledge of the system to a new person. While normally the actor may only want to visualize exceptions, in this case visualization of the structure is required in order to communicate his system knowledge. The defining concepts of a system structure will be discussed in the next section. It may be noted here, though, that several types of contextual information may be considered for a device, among which the following:

- **Aggregation context:** An example of this is an 'engine', that is usually contained in a vehicle.
- **Geographical context:** Many devices do not have a fixed position, but are constantly moving around (airplanes and boats for example). For these and fixed devices as well, the geographical context may provide valuable information.

![Illustration 3.1.5: Example of use of different contexts](image)

- **Spatial context:** The performance of some devices depends on the physical layout of the device. The performance of a package sorting machine, for example, is directly related to the layout and logic of the conveyors it consists of. That context could be made available by providing spatial information about the machine, for example in the form of a three dimensional representation.
- **Organizational context:** The responsibility for maintenance of devices and the processes they are involved in, is always delegated to some organizational unit. Insight in this delegation structure may enable actors to contact the right people in case of unsatisfactory performance of a device.
Illustration 3.1.5 shows a possible context transition from the notification of the failure of a single device (an engine in this case) to a geographical position of its container (the airplane moving over the USA). That information enables for example the determination of the nearest emergency landing spot. The illustration also shows the transitions between the different structural views. In a real-life case, these transitions would be triggered by the actor, who in this way effectively browses the available context. Such a browsing mechanism should therefore be supported by the V&C system. Alongside these ‘physical’ contexts, the actor should also be able to visualize some more abstract contexts, like business processes and interactions. This is discussed in the next section.

Richness of concepts

The most basic concept supported by the architecture is that of (active) objects. On top of that concept, the basic architecture defines basic components like the virtual devices, filters, and virtual actors. When these concepts are compared to the ones mentioned in, the fact that concepts like tasks & processes are missing attracts attention. These tasks and processes are often the actual drivers of the behavior demonstrated by the devices and are crucial in the conceptualization of a business system. Illustration 3.1.6, inspired by [DIETZ96], shows for example a social (business) process that results in a physical movement of a truck. The social process consists of a series of social interactions: actor A1 starts the interaction by requesting the delivery of goods.

Consequently, actor A2 accepts and executes the request (2). This social process triggers the physical movement of the truck. The maximum time the truck needs to move from warehouse two (W2) to warehouse one (W1) is determined by physical and social boundaries. The truck has a maximum speed (physical boundary) and a maximum allowed speed (social boundary). The social process ends when actor A2 declares the result to actor A1. Although an actor might deduce these social and physical patterns from correlations between and distributions of variables, the concepts of social and physical processes would dramatically increase the understandability of the system. In addition, processes and tasks may prove to be the natural way for an

15 In a real-life situation a payment transaction will be triggered as well.
actor to browse a business system.

Although the architecture does distinguish between a social (compare virtual actor) and object reality (compare virtual devices), the architecture does not yet contain a 'process' component. It is, however, not impossible do define the 'process' concept in an object-oriented way. In the meantime it may be concluded that the architecture will not be fully able to support decision-making for business systems, while 'processes' and other concepts that define business systems are not supported by the architecture.

Network visualization and presentation

In cases described thus far, devices and actors have been regarded as autonomous, separate entities, that may incidentally appear in the same context (compare the gasstation view: Illustration 3.1.4 on page 38). As mentioned in the previous section, however, devices and actors in a business system have many relations with other components in the network. These relations may be physical or social in nature.

An example of a physical relation is the decrease of the gas level of the central reservoir at a gasstation, when a customer gets gas from a gaspump (see Illustration 3.1.7). This may seem trivial, but awareness of this relation enables an actor to deduce whether an observed decrease of the reservoir's gaslevel should be considered normal, or should be considered as a leak.

An example of a social relation is the onfollowing order of gas of the gasstation manager. This creates a relationship of mutual commitments and expectations between the gasstations owner and gas provider. Awareness of this kind of relationship enables the actors to keep track of the progress of the transaction thus created.

A final example regards the data communication network (like LAN or WAN) connecting devices. The system administrator responsible for the status of such a communication network will be interested in these device connections.

These examples illustrate that eventually the actors responsible for the devices determine the types of relations of interest. However, regardless of the type, addition of relations to a system of devices will result in a networked system topology. The V&C system should support visualization of these networks by offering tools to declutter the network visualization (compare to the 'presentation' function of the visualization system on page 329). In this way the actor would
obtain a rich system representation, that’s easy to browse.

**Specification phase**

During the specification phase, the actor specifies the parameters for the conceptual model derived in the previous phase. By doing so, an empirical model results. Although the specification phase has been identified as a separate phase, it will usually start during the activity of conceptualization. This is because the significance and relevance of concepts is related to the amount and structure of the objects falling under it. In constructing the conceptual model, an actor will therefore concurrently partly specify the model by inspecting the objects that fall under the conceptual model. The resulting model, however, will only be globally specified. Additional specification of relevant relations and trends is still needed. The way the V&C system could contribute to this will be discussed in the paragraphs below.

**Measuring trends**

The actor may want to abstract from certain system behavior, by specifying models that describe and predict such behavior. This prevents that each individual event needs to be specified. Usually such a model consists of parameters that need to be estimated, in combination with a (statistical) error estimation. The architecture could support this type of abstract specification by enabling the use of a model component. Such a component should contain the model parameters and estimation methods, in combination with visualization components. The model component should be able to operate in two modes: measurement mode and prediction mode. During measurement mode, the model component uses incoming information to calibrate the model-parameters. Illustration 3.1.8 presents an example of the measurement activity. In this example a trend is measured for the average amount of fuel consumption. To do so, a filter has been instantiated that monitors the amount of gas delivered by the gaspump. This filter then calculates the amount of fuel pumped per time unit (for example per hour). The model component plots the resulting information (blue dots) and adjusts its model parameters to accommodate the new information. This is visualized by the red line. The result is a model that predicts the gas delivery as function
of the time of day. When a satisfactory error margin has been achieved, the actor may decide to make the model persistent for future use. This is illustrated in the example by the model database.

**Using model components**

The use of model components does not take place during the specification phase, but rather during the evaluation phase, in particular for management by exception. However, in order to use a model component, the actor should have specified the parameters beforehand, whether by measurement via the architecture, or by other means.

To use a model component for management by exception, it should operate in prediction mode. In this mode, incoming events are compared to the model prediction. When the events deviate significantly from the prediction, the model should emit a notification event, that activates the model’s prediction mode visualization. This situation is visualized in Illustration 3.1.9. In this case the actor has extracted the model component described in the previous example from his database. The activated visualization now shows the predicted fuel consumption (the blue line) in combination with actual information delivered by the filter (the red line). The actor can now conclude that the gas consumption is exceeding his expectation and act accordingly, for example by ordering additional gas.

**Design phase**

The design phase starts when the actor has finished the specification of the empirical model, by having selected the right objects and having instantiated the right filters and models. During the design phase, this empirical model serves as the basis for alternative designs. After performance prediction of these alternatives, a choice can be made. Although the performance of alternative designs could be predicted by developing brand new models, it would be much cheaper if aspects of the empirical model may be reusable for the design models. It has already been shown in Illustration 2.1.7 on page 21 that when simulation models are used as design model, a large part of the empirical model, including the V&C system components, are reusable. The introduction of simulation as the system under control by the V&C system poses the requirements for additional visualization and control components. Some V&C components relevant for simulated but not for
real devices are for example:

- Startup, initialization and reset controls for the simulation (e.g. not the simulated device).
- Simulation time controls: the actor may wish to control the runspeed of the simulation.
- Scenario management controls: A simulated device also runs in a simulated ‘environment’. Different environment scenarios may be available for the actor to choose from.

Since these controls are offered by many simulations, generic V&C controls may be defined to support them. Translation of the generic control commands generated by these V&C controls into commands for a specific simulation environment may be implemented in the virtual device’s wrapper (see chapter 4).

**Implementation phase**

The next chapter discusses that simulation can be used to implement solutions in an incremental way. A relevant consequence for the V&C system during such an incremental implementation is that it should inform the actor of the nature of the devices being visualized. When an actor gives control commands in the supposition that he’s dealing with a simulated device, while in fact the commands are sent to a real device, this may lead to undesirable consequences. This is especially true if an actor is testing the operational boundaries of a potentially hazardous device, like a chemical reactor.

**Active control components**

Up till now, the requirements described have mainly focussed on visualization of information needed for decisionmaking. A basic aspect of decisionmaking, however, is to use retrieved information to act on the system, with some purpose in mind. In Illustration 3.1 on page 28, therefore passive control buttons are shown, that enable an actor to manually perform actions on the system. However, some type of operational decisions are suitable for active (e.g. automatic) control. This may be illustrated by a fictitious example derived from the EGas case. Let us say that a maintenance organization is responsible for the proper functioning of the gaspumps of the gasstations. These devices do not break down suddenly, but usually ‘announce’ upcoming failure by higher electricity or oil use. Lowering pump wear by decreasing pump speed may in that case lengthen the time till failure of the pump. This idea of relating pumpspeed to electricity use, might
turn out to become a standard policy. With thousands of gaspumps under control, it would become very hard though, to implement this policy manually for each gaspump. Instead, this actor may want to be able to define this policy once, and let the system automatically trigger the gaspump’s pumpspeed Business Rule when information on high electricity use is received. This idea is illustrated in Illustration 3.1.10. It shows an active control component that actually listens for events arriving from a filter and consequently triggers the Business Rule component, is located in the Virtual Actor component. This makes sure, that the control component will always function, even if the real actor is not connected to the system (by use of a GUI panel). It is also a logical location, because in this way, the active control component does not act directly, but only by means of a Virtual Actor component, that itself acts on behalf of the real actor. In this way, an actor will always be responsible for the behaviour of the active control component.

The illustration also introduces a GUI panel, that enables the actor to graphically define the logic of the active control component. Such a GUI panel should be able to let the actor logically combine the events arriving from filters and business rules that he is currently subscribed to. The flow-like layout is used to clearly indicate the possible decision outcomes and condition checks. It may also be noted that active control components might first be tested in a simulated environment, as will be introduced in the next chapter.

The active control component as presented in this section, has not been extensively researched during the project. Many questions regarding its use and design should be researched, like:

- **Actor in the loop.** What functionalities should be provided to the actor to stay involved in the process controlled by an active control component?
- **Dedicated control systems.** Currently, dedicated control systems are in place, planning and guarding essential processes of organizations. What are the organization’s experiences with such systems? Do they perform as desired, or do they lack performance or flexibility?

### 3.1.3 Technical Requirements

Based on the actor requirements, technical requirements have been identified that are to be met by an implemented V&C system. Discussed are architecture reliability, distributed component delivery, platform independence, runtime component customization and finally needs for standardization.

#### Architecture reliability issues

The architecture has been developed with a ‘push’-strategy in mind. This means that the architecture should notify the actor if things of interest happen. This relieves the actor from querying the system periodically. In the context of visualization this implies that the visualization components
should be updated as soon as events are generated by the virtual devices. In this sense, the update process can be qualified as 'real-time', since a best possible effort is made to deliver fresh events to the interested actors. However, since the architecture has also been developed to operate on the Internet (see [JACOBS01], it will be inevitable that information is delayed during the transport over the network. This also causes information arriving from different places to arrive out of order. Another source for erroneous information visualization are the device's hardware sensors that ultimately generate the events delivered by the architecture. These sensors are sensitive devices and may be error-prone. If this is the case, the visualization components will again show erroneous information to the actor. It will depend on the business case whether these effects will seriously affect the usability of the V&C system. Two strategies can be used to prevent these undesirable consequences. Measures may be taken to guarantee the correct functioning by guaranteeing network bandwidth and sensor operations (for example by installing multiple sensors). Another strategy is to quantify and display the reliability of the visualization readings. This can for example be done by measuring network delay. Reliability of sensor operations may be quantified by letting the wrapper component perform 'sign-of-life' tests on the sensors. In the end it will be the actor’s judgment whether or not the information provided by the V&C system is deemed reliable.

**Distributed component delivery**

As shown in Illustration 3.1.5 on page 39, visualization components may be very specific for a certain device. This is also demonstrated in Illustration 3.1.8 and Illustration 3.1.9, which show visualization for a model component. This visualization is specific for the type of model component used. It may be recalled that the architecture is developed to support decisionmaking, by making information available in a network of information producing devices, like transport vehicles, production machines and computer systems. There are many of such devices available in an organization, which are of various types. Supplying an actor with all the V&C components needed to deal with these devices is therefore not practical for two reasons:

- It would result in a very large amount of objects to be stored locally with the actor.
- Administration of V&C components would become hard, since an update of such a component would require updating each actor’s local copy of the V&C components.

An alternative is to store all components centrally in one large V&C component repository. This would leverage the administration burden. However, it might become common practice to outsource administration of V&C components in the future, perhaps by specialized companies, perhaps by the device manufacturer. Therefore, a distributed approach has been chosen, in which each component can specify where its V&C components are to be found. This approach still
allows for a centralized approach, while it allows for specific administration strategies as well.

**Platform independence**

The actors using the architecture, will use various types of devices, differing both in Operating System platform (Windows vs. Solaris) as well as in technical capabilities (a handheld device vs. a multi-processor server). This implies that the application implementing the V&C system, that after all runs locally on an actor’s platform, should be able to run on any platform. It also implies that the V&C components delivered by the architecture should match with the technical capabilities of the platform. For a handheld device, for example, a simple text component should be returned, while a full three-dimensional visualization component may be returned for an actor behind a Sun Blade 1000 workstation.

**Runtime customization**

Components delivered by architecture are designed to meet the actor’s expectations. However, not every actor requirement will have been anticipated. The actor should therefore be able to customize or replace the panels and components delivered to him. For the visualization of a model component, for example, an actor may want to influence the update speed of the prediction.

**Standardization**

One of the objectives for the development of the architecture is to leverage the actor’s ability for decision-making. It has also been identified that the main decision-making area considered is the area of business processes. These business processes more and more take place in the context of extensive, global supply chains. Decision-making regarding business processes therefore more and more cross organizational boundaries, which results in organizational and technological complications. The architecture has been developed to support this area of decision-making. It should therefore be deployable over a large area across many organizations. This will only be possible when key technological interfaces have been defined in a way that the architecture is indeed usable for all these organizations. The interfaces of the V&C system that most likely need to be standardized are the following:

- **Event model**: At least the most basic event should be standardized, in order to be able to build correct interfaces for components that need to receive them. By using the concept of inheritance, the basic event should be well extensible, in order to support future developments.

- **Topic**: The most important aspect of the topic interface will be the match method, that checks whether a certain object (for example a visualization component) matches with a topic (for example containing of a technical context description). That part of the interface will not need
to change. The *semantics* of the topic interface, however, will require much more maintenance. What aspects, for example, define whether a visualization component may or may not match with a context description of a simple handheld?

- **Visualization components**: It has already been introduced that there exist many types of visualization components, like 3D, audible or 2D types. The way these components receive input from the V&C system, however, may probably implemented using a standard interface. In this way several dedicated visualization component providers may provide visualization components that are usable within the architecture. Although visualization component providers do yet exist (see for example [ILOG01]), these components use dedicated event and component interfaces.

There are some strategies to achieve the goal of widely accepted interfaces. First of all, a community needs to be formed in which the organizations involved may organize the management and updating of the interface specifications. An actual example of such a community development process is the Java Community Process, see [SUN05]. A second way to develop accepted interface is to define interfaces in a generic way, while at the same allowing for customization when needed. Such customization or extension could then easily be performed by the community development. Finally, if these strategies fail, it will usually remain possible to make any interface available by using *middleware*. This, however, degrades the performance and maintainability of the architecture and should therefore be avoided.

### 3.1.4 Design choices

This section reflects on some design choices that have been encountered during the project. Discussed are choices with regard to the event model and the location of some components in the architecture.

**The event model**

The architecture as described thus far, should update the V&C components by delivering *events*, which are notifications that changes have taken place in the architecture. Such change notification may be diverse in nature, ranging from a change of some property value to the notification that some new device has become available. This section discusses the choice between a generic and specific event model, illustrated by the example event models shown in Illustration 3.1.11.

The left example (the 'simple' model) is the most generic event model. The simple Event is able to represent any kind of Event. For each type, simply another value for the 'type' attribute is used. The simple Event is further able to model many kind of value, since any Event value may at least be represented as an array of Objects. So, on first sight, such a model seems very attractive: it is
generic, while the actor will still be able to interpret a printed simple Event. The content of a simple Event representing a PropertyChangeEvent might for example read: source: "carwash1", type: "PropertyChangeEvent", value: "soapLevel", "50", "gallon". An actor is easily able to interpret this information as the notification that the 'soapLevel' property of the device 'carwash1' has become '50' 'gallon'.

However, although a human actor is able to interpret this information, this model ignores the need for further formalization of the event model. This need becomes clear when an actor for example wishes to be notified only of property change events. For the simple Event case, this might be achieved by inspecting the value of the type field. However, without further formalization, different sources may use different type indications (like 'PropertyChange' or 'StateChangeEvent') for the same concept: a notification of a property change. The same argument holds for the sequence in which the value objects are provided by the Event object. It can therefore be concluded that if this event model is merely used to inform an actor of events, it is indeed usable, but as soon as some more operations are required, this model falls short, without further formalization.

The extended Event model indeed further formalizes the simple Event model. It uses the inheritance concept to specify specific Event types that still share a generic definition (from the root class Event). A PropertyChangeEvent has now become a well-defined sub-class of Event, with well known attributes (namely a value, a name and a unit). The example also suggests the definition of other subclasses in the same way, which choice and design should be the result of a standardization process. The inheritance mechanism, however, provides an apt structure for the for-
malization of the results of such a standardization process, a mechanism which lacks in the simple Event model.

The final example shows that a defining a standard Event model is not evident. In this last example, an extra Unit class has been added. This class offers the functionality to provide a conversion factor between two units. When many actors are involved in the definition of the Event model (as may be the case for the architecture. After all, it is developed for use by many organizations), disagreements may develop. Some actors, for instance, may not need such functionality at all, while other actors (for precision devices like aircraft for example) may argue that the precision of the conversion factors is crucial and should be administrated by an international scientific standards organization. It may be concluded that in that case, the right Event model may be overspecified as a standard. Instead, the middle one should be the starting point that might be extended by several actors for specific needs.

The conclusion of this discussion of the Event model is that its design should avoid both under- and overspecification. In addition, the inheritance mechanism has been identified as a suitable way to formalize the event model, since it provides both formalization as well as possibilities to add specific functionalities.

**Location of components in the architecture**

This section discussed two questions regarding the way certain V&C system functionalities should be delegated to architecture components. The section starts with a discussion of the ways the model functionality described on page 42 could be implemented in the architecture. After that, it is discussed how the representation of relations could be implemented in the architecture.

**Model components in the architecture**

Illustration 3.1.12 shows two possible ways to implement the model component in the archite-
ture. For clarity, the connections of the architecture with the underlying virtual devices, as well as an attached V&C system is omitted. Both examples show datastores, that enable an actor to store and retrieve models and filters once they have been defined.

In the situation on the left, the model component is located inside the Virtual Actor component. The administration of the Virtual Actor is the full responsibility of the the real actor it represents. Adding, updating or replacing model components may therefore be easily performed by the real actor. This model also allows for multiple model components 'driven' by the same filter component. A disadvantage of this model, is the fact that the data reduction accomplished by the model component takes place from the Virtual Actor component onwards. This means that a substantial event flow may still be generated between the filter and Virtual Actor component. The weakness of this model is simultaneously the strength of the second model: here the model component is located in the filter component, where a data reduction will be accomplished. Only the exceptions generated by the model component will arrive at the Virtual Actor component. The disadvantage of this model, however, is that the actor won't be able to replace the model component easily, due to the need for administrative rights. In [HISINKMULLER02] it is presented that for security reasons, actors are not allowed to instantiate just any filter component in the architecture. Instead, rights for a specific filter component are to be granted by an Administrator. Basically, replacing the model component implies re-establishing the rights on the filter. This also influences the ability to instantiate multiple model components: each addition requires new rights for the filter component. The main security threat for a model component running within a filter component is the fact that such a component might execute 'harmful' code on the same machine as the filter. Except for not allowing the model component within the filter component, options are to secure the filter component (by disallowing the model component access to resources) or to use only trusted model components. In the latter case, the model database illustrated in Illustration 3.1.12 should be administered by a domain Administrator, not an individual actor.

**Representation of relations between devices**

As discussed on page 41, the V&C system should provide support for visualizing relations between devices. In order to do this, the underlying architecture should offer a representation of the relations in the network, since this would enable the V&C system to dynamically generate the network visualization (compare with the section on Dynamic panel generation, page 36).

The 'model component' introduced on page 42, provides one possibility to represent relations in the system. Another possibility is to introduce a new component on the level of the virtual device, which is illustrated in Illustration 3.1.13. Where the state of a device is represented by a virtual device, synchronized by means of a wrapper component (see chapter 3), relations could be repre-
sent as virtual connection components that are synchronized by means of listener components. A virtual connection component may store several properties of the connection, for example about the protocol being used and statistics. Protocols may be technical (like FTP or HTTP) or social (like a purchase transaction protocol) in nature.

### 3.2 The lab implementation

This section presents the lab implementation of the V&C system. Illustration 3.2.1 presents the aspects of the implementation that will be discussed in this section.

The lab implementation has been developed in the Java programming language, because of some distinct features like platform independence and its fully object-oriented programming model. The intermezzo below therefore provides a short introduction on the features of the Java language.

After that, subsection I will illustrate some Java concepts used in the architecture. Next, the implementation of the Topic component will be discussed. The onfollowing subsection introduces the visualization interface that enables any visualization component to
receive and process incoming information updates (‘events’). After that, the visualization components developed thus far are presented. Finally, the components and protocols developed to store and deliver remote visualization components are discussed.

After this introduction of the architecture, a discussion on not yet implemented features is provided, in order to identify bottle-necks or promising starting points. Not implemented features include: visual browsing capabilities, model components and actions.

**Intermezzo 2: Java Introduced**

Sun Microsystems has introduced the Java programming language in 1991, as a new development language for the Web era. The language is based on C/C++, but has been developed to be a fully object-oriented language in its own right. Java shares some concepts also found in C/C++, like ‘classes’, ‘inheritance’ and ‘private/public access’. However, where C/C++ also provides a low-level programming Application Programming Interface (API), the Java language has been developed to provide only a high level, object-oriented API. As a consequence, a Java programmer for example does not have to worry about (but also has little control over) memory management. Another consequence is that the Java language is based on a more rigorous object-orientation model than C/C++. Where C/C++ still provides all the procedural concepts developed before the arrival of the object orientation paradigm, Java basically limits the developer to use just its object oriented programming model. The way this Java object model has been used to develop in particular the basic architecture components, can be found in [JACOBS01]. A comprehensive coverage of the Java object and programming concepts can be found in [GOSLING00].

Apart from semantic and syntactic differences, there are many other aspects that differentiate Java from C/C++ and other languages. The following and concluding list provides a short overview of additional Java features:

- **Platform independence**: The Java language has been developed as a platform independent programming environment. This has been implemented by introducing an extra compile layer. During development, Java code is compiled into *byte code*. This code is platform-independent, but not directly runnable. When run, it is processed by a platform dependent interpreter, the so-called Java Virtual Machine (JVM). At this time JVMs are among others available for the Microsoft Windows, Solaris, Mac OS X and Linux platforms. In addition, special (e.g. limited) JVMs are available for small hardware devices and even for chipcards.

- **Extensive standard functionality**: From its introduction on, the Java platform has featured an extensive library of standard functionality. Functionalities featured from the beginning is support for Web-browser enabled applications, so-called *applets*, and a basic Graphics User

---

16 It is, however, well possible to use ‘native’ (e.g. C/C++) code with Java. See the Java Native Interface (JNI) for details.
Interface (GUI) API. Functionality has added ever since. At the moment the Java 1.3 Standard Development Kit (SDK) features for example additional GUI support, sound support, extensive network support (including TCP/IP servers and database connectivity), Remote Method Invocation (RMI) and CORBA for object communication across the network. [GOSLING00] introduces some standard packages, further documentation is available at [SUN06]. In addition, the Java 2 Enterprise Edition (J2EE) adds specific functionality for business information systems, including transaction and mail services. Finally, many extensions to the standard platform have been defined by Sun, including advanced support for image processing, 2D and 3D graphics support.

- **Integrated Development Environments (IDEs):** Since the arrival of Java, many integrated development tools have been developed to support it. Basically these environments offer graphical project management functionality, templates for Rapid Application Development, Java language editor support, graphical GUI development and web-publishable API documentation generation (‘javadoc’). Sun’s most recent IDE is the Forte 3.0 release, freely available for download.

- **Integrated security model:** The introduction of Java for the creation of applets, has introduced the security aspect into the Java environment as well. This is because an applet is basically a program with unknown behavior for the Web browser user who is running it. The JVM therefore was designed to restrict the access of applets to system resources. From JDK 1.3 onwards a new security model is introduced, that supports signing of code archives and fine-grained (e.g. class level) system resource access. The way this has been applied during the development of the Narad project is described in [HISSINKMULLER02]. Detailed documentation on the Java security model can be found at [SUN07].

- **Well established community:** After its introduction, Sun experimented with the management of the Java standard by initiating the Java Community Process, in order to provide an access point for external developers, while still being able to control the quality of the standard. The JCP still exists and has been transformed in a more formal, but also more open process, since other organizations are now as well represented in the executive committees.

### 3.2.1 The Topic implementation

The topic implementation has been designed together with P.H.M. Jacobs, to provide an object-oriented, non-database driven query mechanism. It is already illustrated on page , that such a mechanism may better serve the needs of the V&C system than a traditional SQL-based language. On the one hand this means that any object should be apt for this mechanism, while on the other hand the mechanism should also perform well enough to be usable. In order to do this, a
topic with parts as shown in Illustration 3.2.2 has been developed. The topic object is template or example oriented. This means that the topic contains an example instance of the objects that are considered to fall under it. Any object may act as a template and any object may be matched with it. The interface used to request whether an object instance actually falls under the topic object basically consists of a 'match(Object)' method. This method compares the incoming object instance's type with the template's type. Furthermore, special modifiers may have been defined for the topic, that provide detailed criteria for matching. These criteria may for example require that an object's property value equals the template's property value. The topic object will check whether these criteria are met as well. A detailed description of the Topic class can be found in Appendix B, The topic implementation, on page 7 of the appendices. An additional appendix on the topic implementation can be found in [JACOBS01], where some alternative object comparison mechanisms are described (and criticized) as well.

3.2.2 Some Java concepts introduced

As shown in Illustration 3.2.3, several Java concepts have been used as the basis for the V&C system. The starting point for development was indeed to use standard Java components when possible, which provides the following benefits:
• **High level components and interfaces.** The components that have been defined for the Java platform are defined on a high level, therefore hiding technical implementation issues. This allows for fast project development. Since the components are still fully extendible, by using inheritance and composition, a high level of customization can still be achieved.

• **Well documented implementation.** The standard components of the Java platform are well documented, in the Java-Doc format, which provides much insight in the components (inheritance & composition) structure. All the Java documentation is conveniently available at the Sun web-site, see [SUN08].

• **Well organized standardization process.** The Java standard is administrated by the Java Community process. This guarantees that standard extensions will be well tested and, when possible, will still support older Java code. In addition the Java developer’s connection (see [SUN10]) provides a good platform for (security) bug management (e.g. bug reporting & solving), potentially involving the whole developer community in the process.

• **Small clients possible.** Building a project from standard components, may greatly reduce the amount of libraries needed for client deployment, by installing these basic libraries one time at the clients. Currently this is achieved in the popular browsers by installing Sun’s Java Plug-in (see, that installs a standard Java 2 runtime environment at the client. This ends the earlier practice where each browser developer also developed its own Java Virtual Machine.

Potential disadvantages are that the standard implementation offered via the Java platform just doesn’t fulfill the needs for the project or does contain too many bugs. The Java standards, however, are developed to be very generic in nature. Enough possibilities for customization are available. The amount of bugs in the Java reference implementation has greatly been reduced since its inception. It is also already noted that a powerful platform is available for bug management.
The remainder of this section will introduce the Java concepts used for the implementation of the V&C system. These concepts (as illustrated in Illustration 3.2.3) are Java Beans, Java Swing and the Java Jar & URL protocols.

**Java Beans**

As described in the JavaBeans 1.0.1 specification (see [SUN09]), the goal of the JavaBeans API's is:

> 'to define a software component model for Java, so that third party Independent Software Vendors (ISVs) can create and ship Java components that can be composed together into applications by end users.'

So, on a technical level, the goal of the JavaBeans specification is to provide a way to add and customize components during development time. The requirements and solutions provided in the JavaBean API are therefore focused on the development of an Integrated Development Environment (IDE). Behind the scenes, such an IDE should use the patterns and tools provided by the specification to offer pluggable design-time JavaBean functionality. In addition, a GUI should be provided to allow for easy customizing. Many IDE's like Sun's Forte for Java, Borland's JBuilder and IBM's VisualAge indeed provide extensive Bean based development functionality. The JavaBeans can be customized, stored and retrieved, which have been identified as requirements for the V&C system. In the V&C system, however, components should be delivered and customized during **runtime** instead of **design time**. This follows from the requirement that V&S components delivered by the architecture should be customizable for the actor. This complication may pose additional requirements besides those specified for the JavaBeans. However, designing runtime customizable components as JavaBeans will provide them with the JavaBean’s development features as well. These may prove useful when developing models of alternatives during the design phase of the decision-making process.

The specification identifies requirements that are to be met by JavaBeans and provides solutions by standardizing design patterns and by offering bean utilities in the java.beans package. These requirements and solutions will be briefly discussed in the next paragraph. The final paragraph of the section will discuss the consequences of using JavaBean features for runtime customization.

**JavaBean issues and support**

A brief overview of issues related to the JavaBean is given in the list below. Patterns and support offered in the current Java implementation (e.g. SDK 1.3) will be briefly introduced.

- **Introspection.** A key requirement when using plugged-in components, is the ability to find out the functionalities offered by those components. In terms of object-orientation, this means the discovery of an object’s methods and properties.
Java provides a standard low-level introspection mechanism in the `java.lang.reflect` package, which allows for the retrieval of a class’s fields and methods. Obtaining the values of fields, however, is not always possible using this API, since an object’s fields will usually be declared with private access. The Bean specification therefore specifies special class design patterns that allows assumptions on the method-names that return the value of the underlying field. Such a combination of field and access-methods is defined to a property of a bean. This mechanism allows to introspect properties of a bean, without violating the encapsulation principle. This is illustrated in Illustration 3.2.4, which shows an object with an encapsulated (private) field. The value of the field is for the environment only available via calling the method `getField()` and may only be altered by calling the method `setField(value)`. Standardizing the way these access-methods are to be named (which has been done in the Bean specification), has opened up the possibility to automate the introspection of properties. A further advantage is that methods can be synchronized, while individual fields are not. Synchronization makes sure that multiple threads cannot concurrently access an object’s vulnerable data.\(^{17}\)

The bean introspection mechanism is implemented in the `java.beans.Introspector` class.

- **Customization**: Besides learning the property values of a bean, these properties also need to changed during development. The goal is to provide the developer with an easy GUI interface in order to see and change property values. The Bean architecture supports this by providing a special API for invoking the property’s access-methods. In addition, the specification provides definitions needed for the automatic generation of a GUI panel, consisting of special GUI elements related to the type of the property being changed. Appendix C, The dynamic object editor, on page 14 of the appendices describes how these concepts have been used to implement a tool that enables bean customization during runtime.

- **Event handling**: In an object-oriented application, an object is not automatically aware of other objects. Instead, they have to be linked to each other programmatically. A common scheme realizing such links is the process of event handling. In such a process one object generating an event (for example a change of a property value), communicates this event to all

\(^{17}\) See [GOSLING00] for a detailed introduction on threads and synchronization.
objects that have registered themselves as listeners for that event type. The way objects should announce the events they can broadcast, in combination with the way event listeners should be registered, is specified in JavaBeans specification. Again, this enables an IDE to automatically generate the code needed to realize a connection between event source and listener.

- **Design time versus runtime.** Some Java beans need to perform complicated actions during runtime, or may depend on the availability of database or RMI connections. For such beans, it is not recommended that all the actions are performed during design time. The JavaBeans specification therefore defines a mechanism that allows beans to find out whether they run in a design or runtime. They can thus adapt their behavior accordingly.

- **Persistence.** One of the goals of the JavaBeans specification, is to enable a developer to offer and ship components. This requires that the bean developer somehow should be able to package the JavaBeans developed by him, in order to be able to transfer them to buyers. Technically this has been accomplished by the introduction of the serialization mechanism. This mechanism enables a running bean to be copied as a byte array, that can be stored on disk, or sent via the network (see [GOSLING00]). In addition, the Java ARChive (JAR) has been introduced as the standard persistent archive for JavaBeans, see page 61. Recently (since the SDK 1.4 release) a new eXtensible Markup Language (XML)-based persistence mechanism has been added to the JavaBeans architecture. This enables direct generation of a JavaBean’s XML representation and vice versa (see [SUN12]).

- **Specification enhancements.** In addition to the JavaBeans features described in the JavaBeans specification, additional specifications have been defined in the Glasgow definition (see [SUN11]). The specification consists of the Extensible Runtime Containment & Services Protocol and the Activation API. The Extensible Runtime Containment & Services Protocol defines the way JavaBeans should discover services in their runtime environment. The Activation API defines a way of registering and finding document handling JavaBeans (based on the MIME-type document classification).

### Consequences of JavaBean usage in runtime

Many features mentioned in the previous paragraph are applicable during runtime as well as during design-time. However, some complications do arise, which will be briefly introduced below:

- **No customization available:** The mechanism used by IDE’s to generate panels that allow the modify properties, is usually only available during design-time, as IDE-specific functionality. Therefore, a dynamic object editor has been developed, based on the JavaBeans API. Details on this editor can be found in Appendix C, The dynamic object editor on page 14.

- **No dynamic event handling:** the original event handling mechanism described in the para-
graph above, auto-generates code needed to link objects together. This solution works fine when all the event dependencies between objects are well known during design time. The code needed to hook up individual objects, however, is not easily created during runtime, since this actually requires the construction of a new listener class. In the newest SDK release (1.4), this problem has been solved by the introduction of the `java.beans.EventHandler` class (see [SUN12]), that takes care of constructing a new Listener class on behalf of the requesting bean. This results in the possibility to generate event connections during runtime.

- **Need for synchronization.** For any bean, the developer should pay attention to proper synchronization for beans that can be accessed concurrently. With dynamic bean editing, this becomes even more urgent, since the editor operating on the bean will always access the bean concurrently with the 'normal' program threads. Proper measures should be taken to prevent simultaneous data use & editing.18

### Graphics in Java

The Java Swing components are part of the Java Foundation Classes, which contains the basic graphical packages included in the Java 2 platform. Besides the Swing components, the JFC also contains other packages, including the Java 2D API, which provides advanced 2D graphics operations support. The Swing components, however, are the basic graphic components for the graphical Java application. They are themselves extensions of the basic graphic components of earlier Java releases, the so-called Abstract Window Toolkit (AWT).

The AWT is located in the standard `java.awt.*` packages (see [SUN08]). The basic graphics system provided by the AWT package, provides a transparent update mechanism for Java graphical components, that runs independent from the main application.19 This has been realized using Java’s multi-threading concept, see [GOSLING00] for details.

The basic graphics capable component in the AWT is the

---

18 Unfortunately, the V&C components of the lab implementation have not yet been properly synchronized.

19 This implemented by using an instance of `java.awt.EventQueue` (that collects all.awt-events) in combination with an event-dispatching thread, that processes the events in this queue. This thread runs separate from the main application.
'java.awt.Component’ component. Support for the generation and management of containment hierarchies (e.g. visual components containing other visual components) is provided by the 'java.awt.Container’ component. This has been illustrated in Illustration 3.2.5, that provides a Unified Modeling Language (UML, see [FOWLER00]) representation of some main features of the graphical components in Java. Except for support for addition of components, the Container component also provides the possibility to install a LayoutManager component, that applies an automatic layout scheme on the contained components.

The illustration finally introduces the JComponent20, which is the basic graphic component for the Swing components, located in the 'javax.swing.*' packages. The JComponent’s RootPane component adds the possibility to place graphic components in different layers, which priority may be changed dynamically.

Finally, it may be noted that Swing components comply with the JavaBeans specification and may therefore be developed visually in an Integrated Development Environment (IDE).

The JAR and other stream protocols

The Java input and output (I/O) mechanisms provides a standard I/O interface, by using the concept of the data stream. This concept is used to hide the details of the date source and the protocol used to retrieve it. Datastream providers are conceptualized using the Uniform Resource Locator (URL) concept. A URL object contains a protocol, a location and the name of the resource, that may be accessed and retrieved as a stream. This enables the writing of I/O code that is essentially independent of protocol, location and name of the resource being accessed.

In addition, a special type of Java resource has been defined, the so-called Java ARchive (JAR), including a general protocol to access it (the jar protocol, see [SUN08]). Basically a jar file consists of a structured ZIP-file, with additional archive information in a separate Manifest-file. The JAR may be used to provide class-files, imagem, but also serialized Java Beans. In the latter case, the JAR in fact becomes a simple object database, that can be accessed remotely using a standard protocol like HTTP, FTP of FILE. This last feature has been used to implement a simple V&C component storage.

3.2.3 The visualization Interface

The visualization interface defines the way in which the visualization components can be updated and the type of information objects that can be handled by these components. Underlying the interface, there is the way information is to be spreaded over all available visualization components. A visual representation of the interface is shown in Illustration 3.2.6. This shows that the

20 In Swing, the 'paintComponent(Graphics)' method should be overridden to define the graphical behaviour. Overriding the 'paint(Graphics)' method would interfere with Swing’s containment hierarchy updating mechanism.
• A visualization panel. This panel is actually subscribed to the virtual actor in order to receive information updates. The panel (an instance of `javax.swing.JPanel`) contains all the other components, using the standard Swing containment mechanism. The visualization panel broadcasts all events it receives to all components that implement the interface `com.sun.narad.animation.animationcomponents.Updatable`.

• Topics. Many components in the example contain a topic component. The topic component is used by the visualization component to find out whether the received events should be processed or discarded.

• Updatable components. Every component shown in Illustration 3.2.6 implements the 'Updatable' interface, except for the Topic and Virtual Actor components. The 'Updatable' interface specifies that a component is able to receive update events. The interface is illustrated later.

• Animation panels. Animation components may contain one or more animation panels. The panel do implement the 'Updatable' interface, but do not contain a topic themselves. It is the responsibility of the Visual Component they are contained in to make sure the right events are processed. The animation panels actually implement the visual behaviour of a Visualization Component. They are shown in Illustration 3.2.6 as the graphs and level displays.

The broadcast mechanism
It has been shown that a typical Visualization Panel may contain several components that should update upon receival of an incoming event. The mechanism introduced to implement this, is a broadcast mechanism. This has been chosen, since each Swing component automatically maintains a list of components it contains. Using this list to broadcast events resulted in a straightfor-
ward implementation. The mechanism functions as follows: Upon receipt of an event, the visualization panel delivers the event one by one to each contained component that implements the ‘Updatable’ interface. Each Visualization Component receiving an event will match the event with its Topic object. When matching, it will on its turn does the same: it broadcasts the event to all contained components that implements ‘Updatable’. The animation panels, finally, will process the events and update their visual appearance.

*The ‘Updatable’ interface*

The ‘Updatable’ interface is the main entry point for events delivered by the architecture. The code is shown below. It contains one method that allows an object implementing the interface to receive any kind of Event. The HandlingException is thrown in two situations: the Event type may be unknown, or the information contained in the Event does not match with the topic contained in the implementing Visualization.

```java
package com.sun.narad.animation.animationcomponents;

import com.sun.narad.event.Event;

public interface Updatable
{
    public void update(Event evt) throws HandlingException;
}
```

*The event model*

All event classes are located in the ‘com.sun.narad.event’ package. The event model consists of a basic root Event class, of which many subclasses may inherit. It may be noted that the ‘Updatable’ is designed to pass any Event. When Events are passed that are unknown to the receiving component, a HandlingException will be thrown. However, this won’t affect the functioning of the system.

The basic Event class contains the *source* and *destination* objects. This information allows for routing via several third party event routers. It also contains a *category* property, that defines the semantic meaning of an event. In this way a PropertyChangeEvent may represent an update *notification* (category = Event.NOTIFICATION) or a *command* to update (category = Event.COM-MAND).
The basic Event\textsuperscript{21} model may be extended in many ways, in order to support a wide varieties of information updates. An overview of the currently implemented inheritance tree in shown in Illustration 4.2.2 on page 103.

\textit{Standardization}

It has been mentioned before, that in order to implement the architecture on a large scale, some parts of the architecture should be standardized. This is especially true for the 'Event' and 'Updatable' classes.

In the current implementation, the event listening mechanism may be qualified as an \textit{implicit} listening mechanism, meaning that when a certain interface is implemented (here: 'Updatable'), this \textit{automatically} implies that the component wants to be notified of events. This is only possible when this interface is well known. An alternative solution would be to implement an \textit{explicit} listening mechanism. In this case, each component would subscribe itself to its parent with a custom listening object. In this case, however, this would imply a double registration, since this explicit event registration comes on top of the component registration standard provided by a Swing component. Finally, this last mechanism still wouldn't work if the Event class is not standardized. At least a Visualization Component should subscribe to a certain Event type. If this is not standardized, it will be hard to implement a architecture correctly. Should it for example broadcast two separate types of PropertyChangeEvents, since some components can handle only one type? Such problems would undoubtedly require extra layers of middleware, degrading performance and maintainability. Therefore, if possible, the 'Updatable' interface and 'Event' class should kept standardized.

\subsection*{3.2.4 Visualization components developed}

For the lab implementation, two basic visualization components, namely the StateAnimationComponent and the VarAnimationComponent, were developed. This section will first introduce the general building blocks they consist of. After that, both components are described individually. They can be found in the package 'com.sun.narad.animation.animationcomponents'.

\textbf{Generic animation components}

During the development of the animation components, it became clear that the event broadcasting mechanism, introduced in the previous section, could be considered generic for any animation component. Besides that, it became clear that the visual animation functionalities could be broken down into smaller, reusable components.

\footnote{\textsuperscript{21} The \texttt{java.util EventObject} has not been used, since it defined the 'source' attribute as transient. As a result the source identity will be lost when the EventObject is serialized (during RMI or filestreaming for example).}
The generic animation component is located in the 'com.sun.narad.animation.animationcomponents' package (as are the other animation components). As shown in Illustration 3.2.7 it extends from the basic Swing component 'javax.swing.JPanel'. The 'getComponents' methods illustrated the way the AnimationComponent is able to retrieve a list of contained visual components. The AnimationComponent supports the event broadcast mechanism by allowing a topic to be set. This topic is used when events are processed via the 'update' method. Both methods can throw exceptions. A 'HandlingException' will be thrown when an incoming event does not match with the animation component's topic. A 'TopicNotSupportedException' will be thrown when it is attempted to assign a topic instance that does not contain an instance of 'com.sun.narad.Event' as template.
In addition, some standard panels have been developed, that each have a specific visualization behaviour. These panels are located in the 'com.sun.narad.animation.animationcomponents.panels' package. Illustration 3.2.8 illustrates a typical animation panel design. Some panels implement the 'Updatable' interface. They are automatically updated when they're contained within an instance of AnimationComponent. Usually a panel will override Swing's 'paintcomponent' method, in order to provide custom visualization. The types of panels developed are:

- **ImagePanel.** An ImagePanel is able to display an image, given an image name and a URL location of a Java ARchive (JAR) containing the image.
- **ContextLabelPanel.** Displays the identity and type of the source of the incoming event.
- **LevelAnimationPanel:** Transforms a numerical value into a graphical 'level' representation.
- **StateAnimationPanel:** Contains a link to a background image, combined with a set of states definitions and image links. The panel always displays the background image and displays an image complying with the current state on top. Making such images available in the transparent GIF (GIF87a) format reduces update processing required. The StateAnimationPanel extends from the ImagePanel to reuse its image drawing functionality.

More details are available from the JavaDoc documentation available on the accompanying CD.

**StateAnimationComponent**

The StateAnimationComponent provides a graphical representation of the discrete state of a device. It does so by providing a background image and a list of states. As soon as an event is received containing a state for which an additional state image is available, this state image will be drawn. Additional information, regarding the source of the event and a textual representation of the current state is shown as well.

The StateAnimationComponent extends from the standard AnimationComponent and consists furthermore of a ContextLabelPanel and a StateAnimationPanel. These panels finally implement the StateAnimationComponent’s visual behaviour.

The StateAnimationComponent may (like any Bean) be customized during runtime using the Dynamic Bean Editor. See Appendix C for details.

**VarAnimationComponent**

The VarAnimationComponent provides a graphical 'level' representation of some numerical value. The level is mapped between customizable minimum and maximum values, resulting in a
visualization of the relative value of some property value. The component offers the possibility to show a horizontal level display. Another feature of the component is that the minimum and maximum values of the scale may be updated automatically when values are received that are out of bounds.

The 'LevelAnimationComponent' extends from the standard 'AnimationComponent' and consists of several panels, namely a 'LevelAnimationPanel' and a 'ContextLabelPanel'. In addition, the 'LevelAnimationPanel' may be customized during runtime, enabling an actor to change colours, display orientation and the minimum & maximum values. Customization takes place via the Dynamic Bean Editor. See Appendix C for details.

3.2.5 Delivering remote objects

The lab implementation supports the dynamic generation of visualization panels, based on the actual interests of the actor. Illustration 3.2.11 shows the components and process involved. First, the distributed nature of the panel generation will be recapitulated. After that, the components involved will be introduced, followed by a description of the actual generation process. Finally, the status of the current implementation will be discussed.

Since the architecture has been designed with a large-scale implementation in mind, a distributed way of component storage & delivery has been made a technical requirement (see §3.1.3 on page 46). In this perspective, each basic architecture component (like virtual device and filter) should be responsible for the visualization and control components available for it. It will still remain possible to install a more centralized repository, to which the architecture components can delegate their responsibility for the V&C components.

Component overview

The components involved basically have two functions: to retrieve components from the underlying architecture components and to combine these into a generated panel. The components developed to realize this approach are the following:

- **PanelServer.** The panelserver component is located in the Virtual Actor component and represents the actor's main point of access to underlying V&C components. It implements the 'PanelServerInterface', that only contains one method, used to generate a panel with the appropriate V&C components, given a set of TopicInterfaces. This set of TopicInterfaces is in the current implementation regarded as a list of separate requests. The PanelServer implements the
'java.rmi.Remote' interface and may therefore be deployed as a remote RMI-server. In this way the processing burden related to the panel generation may be transferred to another machine than the machine the Virtual Actor is residing. The 'PanelServer' and 'PanelServerInterface' are rooted in the 'com.sun.narad.animation.-panelserver' package.

- JarAnimationComponentServer. This is a specific implementation of the 'AnimationComponentServerInterface', which is able to retrieve serialized components from a JAR-file (see the introduction on the JAR protocol on page 61). In this way, the V&C component administrator is able to make fully configured components available. The implementation implements

Illustration 3.2.11: The panel generation process

Illustration 3.2.12: Compents for distributed V&C component delivery
the `java.rmi.Remote` interface and may therefore be deployed on a separate machine. The functionalities of the implementation are defined in the `AnimationComponentServerInterface`, that defines several methods to request V&C components. The basic scheme is again to return a collection of components given a set of topics. These components, at the moment, are defined to be general visualizable components (e.g. a `java.awt.Component`). This implies that it is not required to return dynamic (e.g. updatable) visualization components.

The relationships between class and interface are further illustrated in Illustration 3.2.12.

### The panel generation process

Illustration 3.2.11 illustrates the components just introduced and the process that is performed when retrieving components from the architecture. This paragraph will illustrate this process by following the steps one by one.

1. The process starts when an actor has transformed his information needs into a set of topics. Currently, these topics are constructed by a special GUI tool, the `TopicSelectDialog`. This tool provides the actor with options to create a template instance, to set its properties and to specify extra operators. However, this proves to be a rather time-consuming method. Also, the querying power is limited to a selection of events of interest that are already delivered by instantiated filters. A better, more intuitive, topic construction process may be developed, when *browsing* of the system structure has become possible. Visualization of the system structure, however, requires implementation of the concepts of *context* (see page 38) and *network* visualization (see page 41). The underlying topic mechanism may still prove to be useful, given its powerful object-oriented query abilities.

2. Given the topic constructed in step 1, the PanelServer component will query the Virtual Actor component for a list of components that it is currently subscribed to. The request is forwarded

---

22 Rooted in the package `com.sun.narad.animation.panelserver.topicselector`
to all these components. Furthermore, an instance of 'ListeningJPanel' (rooted in 'com.sun.narad.animation.panelserver') is constructed.

3. In the example shown, the request is forwarded to the filter component that the Virtual Actor is subscribed to. Its ComponentServer forwards this request again, since this filter apparently has not defined its own V&C components.

4. The request has reached the lowest level in the architecture: the Virtual Devices. Both Virtual Devices contain an instance of ComponentServerInterface, that’s able to extract V&C components from a database (in the implementation that is a JAR-archive). These components are also able to find the location to the proper data-source and now request these components from the datasource.

5. The V&C components are being retrieved from the datasource and a matching process takes place, in which the components are matched with the Topics that are part of the component request.

6. Matching V&C components are returned to the requester (here: the filter’s ComponentServer), after initialization of their topics.

7. Matching V&C components are returned to the requester (here: the PanelServer component).

8. The PanelServer component adds the returned V&C component to the panel created earlier. In the current implementation, they are arranged using a standard layout. It should, however, become possible to define the context in which the returned V&C components are to be visualized. Based on special attributes, a position within that context should be calculated (compare with the example shown on page 38, where the airplane’s position is calculated by mapping his coordinates to a geographic context). In addition, the panel is subscribed for all the contained topics with the Virtual Actor. Finally the now initialized panel is returned to the V&C system, that adds and shows the panel.

The thus implemented panel generation process allows for a distributed administration strategy for V&C components. Furthermore, the details on type of storage used for the components is hidden in the implementation of the 'ComponentServerInterface'. This allows for storage of components in JAR’s, but also for dynamic object creation based on a database record or textfile. So in the current implementation, a distributed approach to component management has been realized. In addition, these components are customizable during runtime by use of the Dynamic Bean Editor.

### 3.2.6 Discussion on not yet implemented features

This section provides an overview of features that have been identified as requirements, but have
not yet been implemented. Besides a mere observation, this section also wants to provide starting points for future implementation. The features discussed are: browsing capabilities, model components, actions and silent visualization.

**Browsing capabilities**

The current implementation enables an actor to retrieve V&C components from (filter) components that he is already subscribed to. In this way, an actor actually visualizes a part of the total information flow that reaches him (via the Virtual Actor component). A more browsing-like type of visualization, in which an actor can determine his interest iteratively has not yet been implemented. Still, it will be shown in this section that many concepts needed to implement browsing capability are already available in the architecture.

It may be noted that browsing requires another type of information source then when an already available information flow is filtered. Browsing requires that some component is available that can gather information about the system structure, given a certain domain\(^{23}\). Such a component is already available in the framwork, namely as the Jini Lookup Service. This component is a cornerstone of the Jini architecture, which use in the architecture is described in detail.

\(^{23}\) Without a domain mechanism, the browsing action would automatically trigger information gathering on all devices available in the system, resulting in a (too) large amount of information.
in [JACOBS01]. Basically, the Jini architecture allows for dynamic discovery of components in a network, by using a broadcast based protocol. As a result, the Jini Lookup Service is always informed on the components available. This is illustrated in Illustration 3.2.14.

Using the basic information available, the LookUpService is able to provide at least a textual list of components that are available in the system. Such a list is initially static in nature, generated once by a topic request. By using the Virtual Actor component as a common information repository, however, it becomes possible to link information arriving from the LookUpService with information on actual filter instantiations. By combining this information, the browse display can show components for which filters have already been instantiated. This is illustrated in example by the red line in the text display. The gray lines are static, thus just retrieved once. Such a visualization can also function as a user interface for the instantiation of filters. This capability has already been implemented in the demo described in [HISSINKMULLER02].

Additional browser capability may be realized by developing a visual browser. Illustration 3.1.11 shows a possible visualization. It has been observed (see page 38) that visualization of components in this way requires a context, functioning as an ordering mechanism for the components. Therefore a ContextProvider component is introduced, that is able to position components by inspecting some specific property values. The example shows a geographical ContextProvider, that visualizes components based on a position property, for example in GPS coordinates. Again, the browser distinguishes between components that have an additional filter component attached (the red coloured component) and other components. These other component’s positions are retrieved one time, thus not updated dynamically.

The visualization constructed in this way is limited to the components that can be retrieved by the LookUpService component. In the current implementation this is therefore limited to VirtualDevice, Filter, Virtual Actor and BusinessRule components. However, ideas have been presented on the development of concepts like processes and relations (see pages 40 and 41).

Finally, Illustration 3.2.14 suggests the use of Topic objects to support the browsing process. It may be noted that the template objects used when selecting part of the information flow, were of the class Event. Instances of Event, however, cannot be used to specify interest in a static system structure. Instead, other concepts should be used to define the browse interest of the Actor, like ContextDescription, Domain and VirtualDevice. It may be noted that this list is directly related to the amount of concepts that can be retrieved by the architecture: as soon as Processes can be visualized, it should be possible to define a Topic specifying a subset of Process instances. The Topic object itself may still be used, since any Object may be used as a topic template.

Summarizing, it can be concluded that textual browsing is already supported by the architecture, using the basic JINI LookUpService component. This may be extended by visualizing the filters
instantiated by the actor. Finally this browsing mechanism could be extended to become a full visual browser, with support for several contexts. That interface may be used to graphically instantiate filters, resulting in a dynamic system visualization.

**Model components**

The model components discussed on page 42 are not yet implemented. Implementation has two aspects, the development of the component itself and the embedding of the component in the architecture. The latter is easily implemented, since the basic component has been designed to allow for runtime functionality extension by allowing components to be added in runtime (see [JACOBS01] for details. The java.swing.JComponent provides similar functionality). The first is subject for further research, however, given the power of the Java language, in combination with the proven visualization generation, no fundamental obstacles are expected.

**Architecture reliability**

No features regarding architecture reliability has yet been implemented. Technical monitoring of the system, however, may be performed using the basic architecture concepts: by monitoring properties of the relevant devices (e.g. the machine upon which the components are running). The flexibility of Java objects may be used to dynamically transfer objects to other machines, in a similar way as currently V&C components are retrieved.

**Visual control components**

The process used to retrieve control components from the architecture is similar to the process presented for visualization components. The only difference is that control components should be able to place a call to a BusinessRule component, while a visualization component does not actively call the Filter it is subscribed to: it just receives the events. Therefore, no obstacles are expected for the implementation of visual control components in the architecture.

**Silent visualization**

The silent visualization mechanism, as described on page 37, may be easily implemented by using the Swing component’s feature to switch between visual and non-visual mode (see javax.swing.JComponent)

**Simulation control visualization**

This is related to the mechanism used to generate simulations, this will be discussed in the next chapter. Since automatic simulation generation has not yet been realized, development of specific simulation control visualization may be postponed.
### 3.3 Conclusions & recommendations

This chapter has presented the research performed on the Visualization and Control (V&C) system. Two objectives have been the starting point for the development of the V&C system:

- The objective to leverage Sun’s ability to demonstrate the workings of their solutions to their customers
- The objective to leverage the ability to use information procuced in a network of devices for decisionmaking, particularly in behalf of actors responsible for these devices and the business processes they support.

For both objectives, the chapter has identified both functional as well as technical requirements. Also, the lab implementation of the V&C system developed during the project has been presented. In this concluding section, these results will be combined, by addressing the research question as introduced in chapter one. After that, recommendation for further research and development are provided, based on the former results.

#### 3.3.1 The research question addressed

This section will address the final research question as posed in the Introduction chapter for the Visualization and Control system:

> How should the basic architecture, that makes raw information in a network of distributed devices accessible, be extended with a visualization & control system and simulation capabilities in order to meet its objectives of demonstration and decisionmaking?

The research question will be addressed by summarizing and combining the requirements and implementations as discussed in the chapter. The requirements identified specify the implicit ‘what’ question present in the research question. The combination with the lab implementation answers the ‘how’ question, e.g. how the requirements are realized in the lab implementation. It is assumed that the way the requirements have been realized in the lab implementation sufficiently answer the research question, notwithstanding the fact that the lab implementation may not be the only possible implementation meeting the requirements.

The functional requirements have been identified for the two objectives. In addition, technical requirements have been identified as well. This has therefore resulted in 3 concluding tables, that specify the requirements and either indicate where it has been realized in the lab implementation, or indicate that it has not yet been realized. Requirements that have not yet been realized are provided with a comment on the feasibility of additional implementation.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>A need for selection</td>
<td>yes</td>
<td>Realized using the Topic object, see Appendix B, The topic implementation</td>
</tr>
<tr>
<td>Coupled, real-time visualization</td>
<td>yes</td>
<td>A inherent feature of the architecture: the visualization is updated dynamically due to subscription to Filter components.</td>
</tr>
<tr>
<td>User interaction</td>
<td>no</td>
<td>Realizable by providing (passive) control components. No implementation obstacles expected, see page 73, architecture support for control demonstrated in [HISSINKMULLER02]</td>
</tr>
<tr>
<td>Meaningful representation</td>
<td>yes</td>
<td>Visualization components have been developed, providing information on a device's state and on the relative value of a quantity (level display), see §3.2.4 on page 64.</td>
</tr>
<tr>
<td>Dynamic panel generation</td>
<td>yes</td>
<td>Realized by use of Java's serialization and JAR capabilities, see §3.2.5 on page 67.</td>
</tr>
</tbody>
</table>

*Table 2: Functional requirement identified for demonstration*

The functional requirements with regard to decisionmaking have been categorized based on the process model for decisionmaking introduced in §3.1.2, Actor requirements for decision-making on page 36.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evaluation phase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>management by exception</td>
<td>no</td>
<td>Realizable by using Java Swing’s visibility features, no implementation obstacles.</td>
</tr>
<tr>
<td>structural overview</td>
<td>no</td>
<td>Realizable by combining current visualization with Jini’s LookUpService component, see browser functionality discussion in §3.2.6 on page 70. No implementation obstacles identified.</td>
</tr>
<tr>
<td>rich concepts</td>
<td>no</td>
<td>A lack of support for business engineering concepts has been identified. Current concepts supported (devices &amp; properties) already allows for support for control processes (like device maintenance). Identification and implementation of new concepts requires research. No fundamental obstacles have been identified. See §3.1.2 on page 40, §3.1.2 on page 41 and §3.2.6 on page 70.</td>
</tr>
<tr>
<td>network visualization &amp; presentation</td>
<td>no</td>
<td>See above</td>
</tr>
<tr>
<td><strong>Specification phase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>support for model components</td>
<td>no</td>
<td>Not yet realized, no obstacles for implementation identified. See §3.1.2 on page 42 and §3.2.6 on page 70.</td>
</tr>
<tr>
<td><strong>Design phase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>additional visualization for simulation</td>
<td>no</td>
<td>Related to the way simulations are to be generated. Automatic simulation generation is yet subject to further research, see next chapter for details.</td>
</tr>
<tr>
<td><strong>Implementation phase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(No requirements identified)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Active control components</strong></td>
<td>no</td>
<td>A possible design direction has been presented in §3.1.2 on page 44. The issue requires further research, at first focusing on actor's needs and requirements for such functionality.</td>
</tr>
</tbody>
</table>

*Table 3: Functional requirements identified for decisionmaking*
The table below presents the conclusions with regard to the technical requirements presented in §3.1.3 on page 45.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>architecture reliability control</td>
<td>no</td>
<td>May be implemented using the architecture to monitor its own machines. See §3.2.6 on 70.</td>
</tr>
<tr>
<td>distributed component delivery</td>
<td>yes</td>
<td>Implemented using Java's serialization features, see §3.2.5 on page 67.</td>
</tr>
<tr>
<td>platform independence</td>
<td>yes</td>
<td>This is a fundamental characteristic of the Java platform.</td>
</tr>
<tr>
<td>runtime customization</td>
<td>yes</td>
<td>Implemented using Java's Bean model in combination with the developed Dynamic Object Editor (see Appendix C, page 14).</td>
</tr>
<tr>
<td>standardization</td>
<td>no</td>
<td>Importance has been recognized, implementation can only take place after a community has been formed and a standardization process has been passed through. See §3.1.3 on page 47.</td>
</tr>
</tbody>
</table>

*Table 4: Technical requirements identified*

It may be concluded that many requirements have already been realized in the lab implementation. For the remaining requirements, some explorations have already been performed. For none of the requirements serious obstacles for implementations have been found. However, in order to meet some requirements, further research & development is required, which will be discussed in the next section.
3.3.2 Recommendations for further research and development

During the chapter, several issues have been identified that require further research and development. This last section recommends and discusses the development of a visual browser, active control components and model components in combination with research on business engineering concepts and standardization.

Visual browser development

The need for a visual overview of the structure and context of the system made available by the architecture, has been introduced in §3.1.2 on page 38. Structural overview provides an actor with a better understanding of the system of interest, enabling him to construct a conceptual model that combines instruments, environmental influences and criteria. It also enables an actor to communicate his system knowledge to another actor. Development of a visual browser would therefore significantly leverage the usability of the Visualization & Control system.

The current status of the visual browser implementation is presented in §3.2.6 on page 70. It is pointed out that many elements needed for an actual implementation are already available. Development should focus on integration of those components already available, in combination with the development of extra components required, in particular a ContextProvider components, which should provide a visualization environment for visualization components. In addition, a Graphic User Interface, providing controls to select for example domain and context, needs to be developed.

Development of active control components

The active control component as presented in this chapter, has not been extensively researched during the project. Many questions regarding its use and design should therefore still be researched, like:

- **Actor in the loop.** What functionalities should be provided to the actor to stay involved in the process controlled by an active control component?

- **Dedicated control systems.** Currently, dedicated control systems are in place, planning and guarding essential processes of organizations. What are the organization's experiences with such systems? Do they perform as desired, or do they lack performance or flexibility?

Sun's customer network may provide a valuable starting point for such a research, which could be conducted by Sun itself, but also by relations in the academical world.

Development of model components

An essential use case of an environment for decisionmaking, is the support for prediction of
trends and system behaviour. In §3.1.2 on page 42 the concept of a Model Component is introduced, that enables calibration and exception generation of trend models. It is also a fundamental enabler for a 'management by exception' style of control, as introduced in §3.1.2 on page 37. §3.2.6 on page 70 describes the implementation status of the model component, recommending development of two aspects: the embedding of the model component in the architecture and the structure of the model component itself. No implementation obstacles are expected.

Research on business engineering concepts
As mentioned in the introduction section on requirements for decisionmaking, the assumed decisionmaking context for the actor is the business system. It has been pointed out that the operation of many devices is related to a company's primary processes. In addition, two sub-areas of decisionmaking have been identified. The first is the control of maintenance processes, in order to maximize up-time of devices. The current implementation already provides useful support for this, by allowing the actor to remotely monitor a device's properties. The second is providing insight in the business process, that is supported by the device, itself. It has been identified in §3.1.2 on page 40 that the current support for device property visualization does not offer enough concepts to successfully describe a business system. Some key business concepts, from the field of Business Engineering, like processes and tasks, are not yet supported. Adding support for such concepts would greatly enhance the architecture's usability for the business oriented actor.

A research on business engineering concepts would at least consist of two parts: the first part is off course to collect the concepts needed to describe the business system. The second part is the translation of these concepts into object descriptions, since the object concept remains the basic concept used in the architecture.

Research options for standardization
The architecture has been developed with a large-scale implementation in mind, since the business systems of today consist of complex chains of organizations all over the world. Although the current implementation is not suitable to be deployed on a large scale (it is a first-try lab implementation after all), the possibility of such an implementation requires attention for the issue of standardization. §3.1.3 on page 47 identifies the aspects of the visualization & control system requiring standardization, in particular the Event model, the Topic model and the visualization interface.

To accomplish standardization of these aspects, identification of organizations involved and definition of a standardization process will be crucial.
Simulation in the architecture

Simulation may be characterized as computer supported modeling used to understand and predict dynamic system behaviour. The descriptive power of these models are constantly being enhanced, driven by the continuing price decrease of computation power. As such, it is a potentially powerful tool to support the decisionmaking model as introduced in Illustration 2.1.6 on page 21. The prediction potential of simulation models may in particular be well-suited to support the design phase of the decisionmaking process, which after all, requires insight in the consequences of different available alternatives. The concepts of simulation are further introduced in Intermezzo 3, on page 81, while the first section of this chapter will present the actor requirements that have been identified for simulation, based on this decisionmaking model. These requirements have again been partly determined by the E-Gas case.

The chapter then continues with the exploration of some design choices available to implement the identified requirements. The design choices and requirements are the basis for an evaluation of the lab implementation, which describes the way simulation has been integrated in the general architecture for distributed visualization & control. Illustration 4.1 shows the resulting relationship between simulation and the basic architecture. It may be noted that the simulation is situated on the same level as the real devices and therefore is not part of the basic architecture. Instead, the basic architecture is used to make the simulation accessible for the actor, in the same way as it does for real devices.

Finally, based on this evaluation, some recommendations are given for future research and development of the presented simulation subsystem.

Intermezzo 3: Simulation Defined

Simulation is a concept that may call up a variety of associations. In this project, however, 'simu-
lation’ has a well defined meaning, based on Shannon’s definition. Shannon defines the concept ‘simulation’ in [SHANNON75] as

\[
\text{"The process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system."}\]

So apparently, simulation is a tool to study systems. In this context a ‘system’, has been defined in [SOL99], as a part of reality that:

- Is deemed relevant in the context of a certain interest or problem.
- During the time of study can be seen as one whole, separated from the rest of the world, which is considered to be the environment of the system.
- Consists of a collection of objects that have attributes, behaviour and relations.

Shannon’s definition does not imply a type of model to be used. In this report, however, it is assumed that a simulation model is a computer model as opposed to a physical model. The main difference between the two is that a computer model abstracts from the physical mechanisms by using mathematical relations to calculate system behaviour instead of measuring outputs of real physical processes\(^{24}\).

In addition, de Vreede e.a. report in [VREDEDE96] that an animation model has more and more become a valuable extension of the simulation model, enabling communication, verification and validation of the simulation model.

In [SOL99] several scenarios are identified in which simulation might be the most feasible or most (cost-)effective way to meet these objectives. Some of these scenarios are:

- The real system of interest does not yet exist and it is expensive, dangerous or impossible to experiment with a physical model of the real system.
- The real system of interest does exist, but experimenting is expensive, dangerous or not possible without disturbance of the real system.
- The behaviour of the real system has to be studied in (de-)accelerated time in order to understand the desired part or aspect of the system behaviour.
- A static mathematical model of the real system has no simple (analytical or numerical) solution.

Finally, there are several subtypes of simulations described in [SOL99]:

\(^{24}\) Consequently, the validity of the computermodel depends heavily on the validity of the underlying mathematical relations
The interactive simulation. This type of simulation allows user-interaction during the run of a simulation model. In this way the decisions of the users will be reflected in the results of the simulation run.

The gaming simulation. This type of simulation is an extension of the interactive simulation. The gaming simulation allows interaction of *multiple* users, with well-defined roles, during a simulation run.

## 4.1 Design requirements

The next section will identify the requirements and opportunities for extending the architecture with simulation, with the decisionmaking model in mind. After that, resulting technical requirements will be presented. The section concludes with a discussion on some design choices encountered during the project.

### 4.1.1 Actor requirements

The role of simulation in the architecture is introduced in Illustration 2.1.6 on page 21, which shows the decisionmaking process model. In this process, simulation primarily contributes to the design and implementation phase. Some scenarios inspired by the EGas case will be used to illustrate the potential benefits of the use of simulation to evaluate solutions for different levels of the oil company operations:

- **High level:** The oil-company presented in the EGas case administers four refineries, where crude oil, supplied by large oil-tankers, is turned into several products, mostly fuels. These refineries are all located at sea-ports, close to the arriving oil-tankers. The total daily refinery production (about 50 million liters of gasoline) is transported to about fifty terminals, using a network of pipelines. These daily operations are well under control, using some dedicated planning systems. However, the management wants to find out whether really fifty terminals are needed. The question is becoming increasingly more relevant, since due to stricter state legislation on plant safety, the operation costs of the gasoline terminals have increased significantly. Their planning system cannot answer this question, since it only operates on the real system. The management considers an experimental shutdown of five terminals as unfeasible, because of safety and personnel considerations. With a simulation model, however, simulating the terminals, pipelines, production and consumption processes the management would be able to safely predict the consequences\(^\text{25}\) of such a reduction of terminals.

- **Medium level:** Every day, the gasoline arriving at the terminals, is distributed among the 23,000 gasstations, by a fleet of several hundreds of trucks (each capable of transporting

---

\(^{25}\) For example effects on peak fill rates of the remaining terminals
around 30,000 liters of gasoline). A dedicated routing system is in place, that schedules the routes of the trucks. It also calculates the performance of this distribution system, which however does not completely satisfy the distribution manager. Although there are many ideas in the distribution organization to improve performance, the management is again reluctant to implement them (by modifying the rules in the routing system), because of the importance of the distribution system: if a gasstation gets out of gas, the brandname may be severely damaged. A simulation model of the terminals, trucks and gasstations may again provide the management with insight in the performance of the proposed solutions, without the risk of endangering day to day operations.

- **Low level**: A gasstation manager notices that his current ordering policy for food and beverages sometimes leaves him empty of stock. However, since many of the foods and beverages sold by him will not keep long, he does not just want to raise order levels. A simulation of the food selling and supply processes would enable him to test several ordering policies.

These scenarios have shown the potential benefit of the use of simulation. Simulation, however, is not the only modeling tool available. Other model tools like spreadsheets or statistical packages also are used to develop descriptive or predicting models. The advantage of using simulation in this framework, however, is the fact that large parts of it may be automatically generated, due to the fact that many devices (terminals, pipes, trucks, carwashes) may already be virtually represented in the system. Model components may be used to support other modeling techniques as well (see page 42).

**Design phase**

During the design phase, alternatives are to be generated and evaluated. The next section illustrates how the architecture could be used to support the automatic generation of simulation models. In addition, two types of simulation use (gaming simulation and distributed simulation) are presented, that pose additional requirements to the architecture and the simulation. Both requirements have not yet been realized in the lab implementation.

*Generating a simulation model*

Illustration 2.1.7 on page 21 shows the basic concept of using the virtual representation of the system to automatically generate a first simulation model. The example shows how a system consisting of some gaspumps is turned into a simulation, which may for examplebe used to evaluate effects of changes in the customer arrivals patterns on the queue lengths and waiting times. Since the architecture is indifferent to the exact nature of the underlying device, a system of transportation devices (like gastrucks) could also have been turned into a simulation model, for example to
evaluate routing performance. The following aspects should be taken care for, in order to successfully generate a simulation model:

- **Object types**: the real devices available on the lowest system level should return as simulated objects in the simulation. For the gaspump example this implies that in the simulation model gaspump objects need to be instantiated, while for the gas truck example, gastruck, terminal and gastank objects should be instantiated. Illustration 4.1.1 shows two possible approaches with regard to object instantiation.

In the first approach, the wrapper component, that provides the communication between the virtual and real device (see next chapter), is replaced with another wrapper component, that merely provides communication with a simulation model. In addition, a gaspump instance has been instantiated within the simulation model. In the second approach, the wrapper component is replaced with an active simulation component, that has autonomous, active behaviour.

In both approaches, the simulated entity receives the same events as the real device, since the Virtual Device component is unaware of the presence of the simulated device. The instantiation in the first approach is more limited and complicated, though, because each simulation environment has its own definition of simulation entities. Furthermore, dynamic generation of entity definitions may not be possible. In the second approach, the active simulation component may basically directly access the properties of the Virtual Device it is contained in. The active simulation component therefore merely consists of the active behaviour of the virtual device. Like visualization components, virtual devices could administrate and instantiate their own active simulation components, using the methods introduced on page 67.

Due to the absence of simulation model specific characteristics, the second approach promises to be easier to standardize and therefore to automate. However, this will require standardization of the active simulation component itself, which, at the moment, is not even yet developed.

Finally, for both approaches it will be required to make a copy of the original virtual device object.

- **Filters & Business Rules**: The Filter and Business Rule components instantiated by an actor on a real system, should preferably be automatically copied and instantiated on the simulated
system (as illustrated in Illustration 2.1.7 on page 21). When these are in place, visualization on the simulated system may automatically generated. After all, the copied Virtual Device components (see previous item) are still aware of the location of their V&C components.

- **Interobject relations:** In a simulation environment, modelling of devices is not enough. In addition, specific relations between simulated devices need to be explicitly modelled. In the middle level EGas scenario, which consists of terminals, gastrucks and gasstations, there are relations between the gastruck, terminal and gasstation. After all, if a gastruck is loaded with gasoline, the gasoline level of the terminal should lower (compare to section 3.1.2 on page 41). While this happens automatically in reality, this won't happen in a simulation model unless additional relations have been defined.

- **Initial state:** The initial state of device and filter properties may automatically be derived from the original Virtual Device and Filter components.

- **Behaviour:** The behaviour of the devices in the real system is determined by the way real devices work. This is illustrated by Illustration 4.1.1: via the wrapper component the device determines the state of the Virtual Device component. When a simulation model is made, this behaviour needs to be modelled as well. In the two approaches shown in Illustration 4.1.1, this is either achieved in a specific simulation environment, or by adding behaviour to an active simulation component. Since the architecture is implemented in Java, the thread mechanism can be used to achieve active behaviour (see [GOSLING00]). The passive behaviour of the component is modelled by reactions on events that it can receive.

It may be concluded that many requirements for the automatic generation of simulation are potentially easily met by the architecture, in particular the initialization of the state of the components, instantiation of the right Filter and Business Rule components and finally the availability of V&C components for the simulated devices. A standardized way to instantiate simulated devices, and behaviour, however, is not yet supported by the architecture. However, in order to realize automatic simulation generation as a fundamental characteristic of the architecture, standard API's to look up, instantiate and initialize simulation components need to be developed.

**Simulation model editing**

In the previous section, how the architecture could support the generation of a simulation model representing a 'snapshot' of the system. However, during the design phase of the decisionmaking model, not only the current system performance is of interest to the actor(s), but rather the performance of alternative system designs. Some possible model modifications required in that case are:

- **Adding object instances.** In the gas distribution example, an actor might want to evaluate the
consequences of additional transport means. He therefore needs to instantiate more transport means than those automatically generated.

- Changing parameters. An actor may want to change parameters of the model, like the maximum load of transportation means.

These aspects have not been researched in this project. However, enabling an actor to modify an automatically generated simulated system, would leverage its usability.

**Gaming simulation**

The architecture (as shown in Illustration 4.1 on page 81) allows multiple actors to access the system, with each actor having assigned its own user-specific view on the system. Also, each actor is able to monitor and act on the system. These two characteristics combined, allow different actors to perform different roles in the system. If that system does not contain real devices, but instead a simulation of these devices, the system effectively becomes a gaming simulation. Illustration 4.1.2 shows a simple gaming simulation setup, in which two devices are simulated. The two actors shown each have monitoring and control options for a single device.

An example of an application of gaming simulation for the EGas case is the production planning of the refineries. The management realizes that unexpected changes in demand at the gasstations, often leads to large oil stocks at the refineries. Since the oil terminals at the seaports are becoming increasingly more expensive, the refinery management searches for solutions to decrease the observed stock impacts of unexpected changes in demand. These solutions primarily regard ordering policies used by the gasstations in times of unexpected decline of demand. Such solutions could be evaluated by a gaming simulation that predicts the consequences of the ordering decisions of actors involved on the stock levels within the supply chain.

**Distributed simulation**

In the scenarios described this far, there was only one simulation system present in the system.
For large scale systems, however, it can be beneficial to be able to divide a large simulation model into submodels running on separate computers. In [FUJIMOTO00] several potential benefits of distributed simulation are stated, among which:

- **Reduced execution time.** A direct result of the fact that the total computer resources available for a distributed simulation are usually significantly higher than those available for a monolithic simulation.

- **Fault tolerance.** In a distributed simulation environment, the system can be built with redundancy. This may, if combined with an appropriate take-over protocol, provide fault tolerance in case an individual simulation fails.

However, simulations in a distributed environment will usually not be able to run completely independent of each other. In [KUHL99], several needs for interaction are distinguished, among which:

- **A need for data sharing.** It is not unusual in a distributed simulation environment that simulated entities need to ‘cross the borders’ of the subsimulation they are contained in. An example of this is the simulation of a moving aircraft in a system of regional airspace control simulations. When the simulated aircraft moves into a new (simulated) airspace control region, the relevant information over the aircraft will somehow have to be transferred to the simulation that models this new airspace.

- **A need for time-synchronization.** A distributed simulation system that runs at full speed, in order to calculate a certain ‘what-if’-scenario, will encounter the fact that certain individual simulations are not able to complete their run as fast as other simulations. In order to prevent consequent errors in the results, the simulations will somehow need to exchange time information during the simulation run.

So, in order to benefit from the potential benefits of distributed simulation, the technical issues regarding time and data synchronization need to be addressed. In practice, solutions for these issues will themselves use up resources. Time synchronization, for example, will likely be addressed by some kind of network communication based synchronization protocol, which can take up a significant part of network bandwidth, thereby preventing the realization of the full potential of distributed simulation.
Implementation phase: ‘soft-commissioning’ and ‘reality in the loop’

The support for real and simulated devices running simultaneously in one system is one of the requirements for ‘soft-commissioning’ and ‘reality in the loop’, as described in [Auinger99]. In this article an architecture is developed with the objective to make the design and testing process of systems more effective and efficient. The authors define the term ‘soft-commissioning’ as a situation in which a real control system controls a simulated system. An example of such a system in the EGas case, is a routing & planning system for the gas transport between terminals and the individual gas stations, that takes place by trucks, pipelines, railcars and marine vessels. The distribution network in the EGas case consists of around fifty terminals, that supply around ten thousand gas stations. Per terminal, around two hundred gas stations need to be supplied. For the daily planning operations, control systems are in place, that schedule and calculate the routes of the transportation means, based on supply and demand information. Given the constant pressure of the oil company’s competition to cut costs, efficient and effective distribution is crucial for the oil company’s success. Also, many different types of planning algorithms may be used by the planning system. Both factors raise a need for simulation, either to evaluate the performance of a new planning system or to optimize the operations of an existing one. Both needs would be addressed by soft-commissioning the control system.

Illustration 4.1.4 shows, however, that the control system design may need to be altered, in order to support easy soft-commissioning. The left situation shows a common control system architecture in the bottom layer: the control system (represented by virtual device 3, a server) directly communicates with the devices fulfilling its information need: positions & gasoline level of transport.
portation means and gasoline levels at the terminals and gas stations. This direct connections, however, make it hard to introduce simulation (and thus soft-commissioning). Although each device is already available as a virtual device, these cannot be used to instantiate a simulation object in the way illustrated in Illustration 4.1.1 on page 85, since the control system would then not be able to communicate with the simulated device. Soft-commissioning in this example is still possible, by developing dedicated simulation objects that are able to communicate with the control system. This, however, would ruin the concept of the virtual device, which has been developed to hide communication and implementation details of the device. The right example presents an alternative control system location in the architecture, that does allow for easy soft-commissioning, since the control system communicates with the devices using the Virtual Device components. To introduce soft-commissioning in this situation only requires the instantiation of the device’s standard simulation components, as introduced in Illustration 4.1.1. These simulation components need to be developed as well, but may be developed just once, without the need to develop dedicated network communication functionality.

In addition, the term ‘Reality in the loop’ may be defined as a situation in which a simulated system is supplied with information from real devices. An simple example of such a system might be a simulation component that models a gasoline tank. One of the properties of such a tank is the volume of the contained gasoline. When a real terminal would be made available in the architecture, by means of a Virtual Device, only a volume measurement device would need to be wrapped to make the volume available. However, the volume measured actually depends on the outside temperature of the terminal: with rising temperature, the gasoline volume in the tank will rise as well. Although this effect could be modelled as well in the simulation, another approach is to adjust the volume using the measurements from a real thermometer (e.g. ‘reality in the loop’). A possible implementation of this situation is illustrated in Illustration 4.1.5, which immediately raises the question, how the simulation is to be supplied with the readings from the thermometer. After all, there is no direct connection between the two. However, indirect communication is possible,
since the thermometer and the simulation are available as Virtual Devices. A direct communication between the simulation’s and the thermometer’s Virtual Devices is considered to be not desirable, as described in [HISSINKMULLER02]. Therefore, the simulation may need to delegate its information needs to the listening actor, represented by the Virtual Actor component. The actor then needs to subscribe to the thermometer’s temperature reading and relay this information back to the simulation. A standard way in which a simulation may notify an actor of its information need yet needs to be developed.
4.1.2 Technical Requirements

In the previous paragraph, several application areas have been identified, that each pose different requirements upon the simulation subsystem. In Table 5 an overview is given of several requirements identified per application area. The requirements are grouped into several requirement categories, that will be discussed in the following paragraphs.

<table>
<thead>
<tr>
<th>Requirement category</th>
<th>Requirement</th>
<th>Soft Commissioning / Hardware in the loop</th>
<th>Gaming simulation</th>
<th>Distributed simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event delivery</td>
<td>Acknowledged delivery</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Known latency</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Event ordering</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Simulation development</td>
<td>Design tools to constrain environmental interaction</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Real-time</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data / time synchronization</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Separate visualization</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Security</td>
<td>Authentication</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Confidentiality</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Integrity</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

*Table 5: Simulation subsystem requirements*
Event delivery

As shown in the illustrations of paragraph 4.1.1, the simulation in the architecture is always connected with other components on the network. Since an asynchronous rather than a synchronous approach is used for the communication (see [JACOBS01]), the necessary exchange of information between these components takes place via event notifications (in the following simply referenced as events). In order for the system to run without errors, a certain quality of service level, with regard to guaranteed event delivery and latency, may have to be met. While the quality of service is a characteristic of the network, the components in the network should be able to monitor the quality of service level, in order to be able to initiate 'emergency procedures' in case the quality of service level drops below an unacceptable value, from a business point of view. It may be noted that these requirements are not only relevant for simulation systems, but for real devices and systems as well (compare for example Illustration 4.1.4). For some aspects of quality of service this will be illustrated below:

- **Acknowledged delivery.** In the case of a system property that’s regularly updated, the loss of one event will not cause damage. However, in a ‘hardware in the loop’ system, the undetected loss of a control event destined to close for example a high-pressure valve could have disastrous results. In this case, the sender of the control event should at least be notified of the send failure. The components should initiate ‘emergency procedures’ when too many losses are experienced, for example by initiating a graceful shutdown.

- **Real-time event processing.** For some of the application areas, the simulation may need to run in real-time, especially to interact with physical components that naturally run in real-time. An example could be the soft-commissioning of a refinery operations safety control system. Real-time in this sense may be defined as ‘the ability of a system to process incoming information at the same rate as it arrives’. In other words, in the real-time receiving system no information queues are formed. An additional aspect of real-time is that the processing time of a unit of information is constrained. According to [BOLLELLA00], it is precisely the time-constrained nature of real-time execution, not the requirement for ‘as-fast-as-possible’ execution, that characterizes real-time computing best. From this it follows, that real-time systems in a distributed environment (like the NARAD architecture) only make sense, when the latency experienced in the communication networks connecting them, also is constrained. Real-time systems should therefore be aware of the latency experienced and initiate 'emergency procedures’ when intolerable latency is experienced on the network.

A distributed simulation system (e.g. not connected with any physical device) does not require

---

27 The Oxford English Dictionary defines an event as ‘something that happens’. In the field of information systems and simulation this ‘happening’ is assumed to have a negligible duration. In case of a control event, it has the meaning of ‘something that should happen’. Finally, in case of a notification event, it has the meaning of ‘something that happened’. 
a maximum event delivery latency, when used to calculate a scenario, since such a system will advance time in a controlled way. Latency in the network will, however, degrade simulation speed.

- Event ordering. Application areas are plausible in which the order of incoming events needs to be unambiguous. Two control events sent by a gasoline pipeline control system arriving out of order (for example one for opening and one for closing a high-pressure valve) can cause disaster! Again, this requirement is therefore not limited to simulations in the architecture. The requirement is, however, more relevant when distributed simulation is used, using 'as-fast-as-possible’ execution, for example to predict long-term impact of changes in the oil company’s gasoline distribution policy. A typical simulation model on a modern laptop is able to generate around ten thousand events per second, during an 'as-fast-as-possible’ simulation execution, which implies an inter-event time of much less than a milli-second. Since variations in network delay in a normal Local Area Network will usually amount some milliseconds, many events will arrive out of order, if each event is sent separately. The only way to prevent this from happening, is to time-stamp the events, in combination with the use of time synchronization protocols (see [FUJIMOTO00]).

For real-time simulation (e.g. when simulation time advances at the same speed as wall-time), and normal devices, a dedicated event ordering mechanism may not be needed. In this situation, after all, the time between causally related events will usually be much greater. An example of causally related events, are the position updates of a transport device. If these events arrive out of order, the trajectory of the transport device will seem very odd. However, a typical transport device may for example only need to update its position once every minute. The inter-event time is therefore much smaller than the average network latency, which may be less than half a second round trip time for a connection between Delft (NL) and Palo Alto (CA). The position update events will therefore arrive in order. Although it is possible that events coming from two independent transport devices arrive out of order, this is not problematic, since the two devices, and thus their position events are not causally related to each other.

**Simulation development directives**

In order to use simulation in the architecture, at least components need to be developed that simulate devices (see Illustration 4.1.1 on page 85). The fact that these simulation components need to run in a distributed environment (e.g. the architecture) impacts the model design. One impact originates from the fact that introducing an *environment* also implies introducing *environmental uncertainty*. This uncertainty results from the fact that the modeler can’t predict beforehand what
the environmental input (e.g. events arriving via the Virtual Device components) on the model will be, while in a single simulation model all the events are created within the model itself. Another impact has to do with the performance of the model: broadcasting state changes to the environment uses up scarce (computer) resources. In order to constrain the environmental uncertainty and extra resource needs, some simulation development directives should be developed to control, for example:

- Sorting and routing of the information received from the environment. The incoming information should be routed to the submodel that’s able to process it. In addition, some form of exception handling mechanism should be available, to prevent instability due to the arrival of incorrect events.

- The kinds of information the model should make available for the environment. This prevents the model from broadcasting all the state changes of the model, resulting in superfluous computer and network resource needs.

- Properties of the model that are allowed to be modified from the environment. This basically limits the inputs that are accepted from the environment and therefore constrains environmental uncertainty. It may be recalled that this is also the main function of the encapsulation principle used in object-orientated programming.

- Different policies on the aforementioned dimensions for different types of environments (for example trusted vs. non-trusted environments). This is less relevant when the simulation is used in the architecture, since security is already explicitly adressed in the architecture (see [HISSINKMULLER02]).

It will be shown in the introduction of the lab implementation that some of these directives have already been implemented, for example in the form of special ’event broadcast’ building blocks. Experience with a real-world implementation of the architecture may however be needed, in order to assess the performance of the developed design directives.

**Simulation execution**

For application areas where (direct or indirect) interaction between simulation models and physical components takes place (soft-commissioning and reality in the loop), a real-time execution of the simulation models involved is required\(^\text{28}\), since the simulated devices should behave as the physical components they represent, including their response times and other time-related behaviour.

Another aspect of simulation execution is the visualization of the simulation’s animation. In most

\(^{28}\) A real-time simulation execution means that the simulation time advances in the same pace as ’wallclock’ (or real) time.
of the application areas described, there can be more than one user (actor) at a time, each with specific needs for visualization and control. This implies that the visualization part of the simulation model should be separate from the simulation model itself, allowing several visualization engines to 'hook up' to the simulation model (as illustrated in Illustration 4.1 on page 81). For a detailed treatment of the way distributed visualization has been implemented in this project, the reader is referred to chapter 3, starting on page 28. It may be noted that it has been assumed in this project that animation of a simulated system for the most part resembles visualization of the real system, the only difference being the source for dynamic behaviour (a simulation vs. real devices). In [VREEDE96], however, it is presented that in addition of the animation of the problem situation, some aspects specific for the simulation model (like the simulation time) should be animated as well. These aspects, in combination with additional components needed to control the simulation, have been considered to justify the development of a separate simulation V&C panel, as introduced in §3.1.2 on page 43.

The final requirement with regard to simulation execution is that of data and time synchronization. This is only a requirement for a distributed simulation system, as mentioned in paragraph 4.1.1 on page 87.

Security

In all application areas described, security play an important role. This is mainly because the architecture has been developed to operate on the World Wide Web, the ultimate example of an insecure environment. The way general security issues are dealt with in the architecture are treated in [HISSINKMULLER02]. Some specific security issues, that have to do with the interaction between simulation (or real device) and the architecture will be discussed in chapter 5, Wrapping devices, starting on page 114. It may be noted, however, that for the systems described in paragraph 4.1.1, security can be a requirement of the highest priority, depending both on the specific application (compare the refinery security control system vs. gaspump monitoring) and the deployment environment of the architecture (compare the public Internet vs. a closed, private network).

4.1.3 Design choices

There are several design dimensions available to meet the requirements stated in the paragraph above. In this paragraph several design dimensions will be discussed in the context of their impact on the requirements, in order to provide an overview of the available design freedom.
Monitoring and controlling event delivery

Communications between distributed nodes on a network takes place via several layers of communication. The Open Systems Interconnection (OSI) model, a reference model standardized by the International Standards Organization (ISO), for example, identifies seven different layers of abstraction, ranging from physical to application level. An overview of several categories of computer network systems can be found in [HALSALL95]. So, given a computer network system, the monitoring and control of event delivery could be implemented in one of several layers. The two extremes are to implement these functions in either on an application level (high level) or in some network layer (low level). Implementation on a high level provides the benefit of independence of the underlying network system. However, implementation on a high level is not always required, since many computer network systems nowadays do provide at least some standard form of reliable service in the network layer.

Since the TCP/IP protocol stack (see [HALSALL95]) is nowadays a very common and popular network protocol stack, it will be discussed in some detail below. TCP stands for Transport Control Protocol and IP stands for the Internet Protocol. IP is a low level protocol, delivering a connectionless, non-reliable communication service between computers on an internet. TCP is positioned on top of IP and delivers a reliable, connection and client/server oriented service between two remote applications. The data transmitted over a TCP connection is distributed by TCP in data segments. The TCP service has the following QOS properties:

- Received segments are error free, by using error detection and resolution
- No duplicate or lost segments, by using flow control.
- Segments\(^\text{29}\) are received in order.
- Acknowledged transmission: the sender will be notified if a segment is undeliverable.
- Options to send data segments directly (without buffering) or urgent (no flow control)

So TCP seems to be able to meet some of the stated technical requirements: acknowledged and sorted data delivery. However, TCP only sorts data on a particular connection, NOT on concurrent connections. In case multiple components connect concurrently with another component (see Illustration 4.1.6, in which two filters can connect concurrently to one virtual actor component), TCP does not guarantee a delivery order. However, it has been discussed on page 94 that explicit event order monitoring is not always required.

---
\(^{29}\) This guarantees that contents over a particular connection are delivered in order. The final received order of data sent via concurrent connections is NOT guaranteed.
In addition, TCP does not monitor the latency experienced by the communicating components. Finally, in case of a transmission failure, the sending component will be notified of the delivery failure, while the receiving component will remain unaware of the failure. To resolve this undesirable situation, a life-pulse protocol may be used on application level. In such a scheme, components broadcast null-messages with well-known intervals. As soon as a component’s null-messages aren’t received anymore, or not on time, it becomes clear that it is not normally reachable anymore, in which case corrective measures should be taken. When the fact, that TCP / IP has become the standard networking layer, is taken into account, it can be concluded that most of the requirements regarding event delivery may be delegated to the TCP / IP network layer.

Remote event delivery

An important design choice with regard to remote event delivery, is the way in which remote event delivery differs from internal event delivery in the simulation model. Remote events could differ in data structure as well as the processing they undergo. This section presents two different approaches for (remote) event delivery.

One approach is to minimize difference: all events are processed in the same way. This means that internal events, would be 'looped back' into the model, using the same routing process as remote events. The data structure of the events can then be the same: the only difference between local or remote events would be the value of the destination address. This situation has been illustrated in Illustration 4.1.7.

The event handling mechanism’s simplicity eases the modeling process and also makes it very
flexible: in a distributed simulation environment, the moving of a submodel towards another simulation would cause minimal alterations in the simulation model. In addition, the event flow may be monitored precisely, since each event will pass through the general event routing system. The fact that any event passes through the general event routing system, also allows for the creation of a ‘proxy’ object in the general event routing submodel, that may automatically route all events destined for submodel B to its new remote address. Even more intelligence could be built into the event routing subsystem, by implementing a *naming service* able to dynamically determine the location of submodels on the network. In both ways, the submodels in simulation A do not even need to be aware of the movement of submodel B.

The disadvantage of this mechanism, however, is the performance loss that results from the processing needed within the general event routing system, which after all needs to process any event in the local model.

Another approach is to allow multiple event handling mechanisms and event data structures. This situation is illustrated in Illustration 4.1.8. In this situation, the remote event routing subsystem can be implemented more lean, since no check for local addresses needs to be performed. Also, local events could be processed faster, since no extra subsystem processing is needed for them. However, movement of submodel B would raise the need to redefine all local event paths to submodel B into remote event paths. The solution of a proxy object or naming service is not available in this situation, since the events do not necessarily pass the routing subsystem. It will be shown in the presentation of the lab implementation, that this last approach has been chosen, considering the unlikeliness of the need for dynamic transfers of simulation (sub)models in the archi-

![Illustration 4.1.8: Approach B: Two event handling processes.](image-url)
tecture and considering the importance of simulation performance, given the real-time requirement for simulation in the architecture.

**Creating real-time simulation**

The realization of real-time simulation has not been pursued in the NARAD project. However, the following aspects and products for real-time realization have been identified:

- A real-time simulation runtime environment should be used, to synchronize simulation time with wall-clock time. An example of such an environment is Arena RT (see [ROCKWELL01]).

- A operating system with real-time capabilities could be used, to guarantee for example a maximum response time between an interrupt and a user process’ response. Windows (see [MS01]), Linux (see [RTLINUX01]) and Solaris (see [SUN01]) all provide support for real-time processes.

**Data / time synchronization**

The interaction protocol used to synchronize data and time between simulations could either be newly defined, or be based on an already existing and supported standard. [FUJIMOTO00] provides a comprehensive background on the field of parallel and distributed simulation systems, which could serve as the basis for a new interaction protocol.

In addition, during the 1990’s the American Department of Defence has developed the High Level Architecture (HLA), to enable the interaction of a multitude of military simulations. HLA provides a rich set of functionality, both for data and time sharing. During the NARAD project, a first literature study has been done for HLA, which is included in Appendix D. However, the performance of HLA in comparison with other synchronization protocols has not been researched during the project.

### 4.2 The lab implementation

For the NARAD project, a first implementation of a simulation model that can interact with the architecture has been developed. This implementation has been included on the project CD, with location `<drive>\Arena\Force_demo`. The implementation has been built in the Arena simulation development environment from Rockwell software (see [ARENA01]). The realized role of the simulation in the reference implementation is best illustrated by Illustration 4.1 on page 81.

In the lab implementation the simulation is connected with the rest of the architecture by means of a virtual device (see [HISSINKMULLER02]) and communicates with this virtual device by means of a wrapper. The wrapper translates the output of the simulation into a form that's under-
standable by the virtual device and vice versa. The workings and requirements of the wrapper will be discussed in more detail in the next chapter. Finally, the simulation communicates with the wrapper using the Common Object Request Broker Architecture (CORBA), a widely accepted standard for distributed object communication maintained by the Object Management Group (OMG, see [OMG01]).

This chapter will first introduce the different components used to communicate between simulation and virtual device. After that, the event model that’s used for the communication will be introduced. On following, the way the remote event handling has been integrated in Arena will be illustrated. Finally, an overview will be given of features that are not yet realized in the reference implementation.

### 4.2.1 Components used between simulation and virtual device

The way the simulation communicates with the corresponding virtual devices is illustrated in more detail in Illustration 4.2.1 on page 101. It is shown that the simulation application (Arena) communicates with the CorbaDeviceWrapper (the wrapper implementation) via several intermediate programs. In the remainder of this chapter, such a program that’s able to execute autonomously, will be called a *component*.

Some components run in the native Windows environment, while others run in the Java Runtime Environment. The components running in the Windows environment are all30 Dynamic Link

---

30 Arena is not a DLL itself, but is linked during runtime with a user DLL (here: the EGasDLL). This DLL is able to com-
Chapter 4  Simulation in the architecture  102

Libraries (DLL), a Windows specific component type (see [MICROSOFT01]). DLLs have capabilities to communicate with other DLLs. However, DLL’s are not able to connect directly with components within the Java Virtual Machine. Therefore, a communication mechanism and a protocol were needed to establish communication between the DLL components and Java. The Common Object Request Broker Architecture (CORBA), a standard for distributed object exchange maintained by the Object Management Group (see [OMG01]) has been used to provide this communication. A detailed overview of CORBA can be found at [CORBA01].

The components and their functions will be introduced below:

- *OmniORB2 & Java ORB*. These components function as Object Request Broker (ORB). ORBs communicate with each other using the Internet Inter ORB Protocol (IIOP), which enables ORBs to establish communication over an internet. This basic communication service is used by the ORBs to transfer (requests for) CORBA objects. However, although the communication between ORBs has been standardized in the IIOP, the communication between application and ORB is not standardized. The (requests for) objects transferred between application and ORB therefore have ORB specific characteristics. To deal with these specific characteristics, the Interface Definition Language (IDL, see [OMG01_2]) has been introduced as a generic interface specification language. It’s up to the ORB to provide a mapping between the IDL specification and the environment of the ORB’s client applications. OmniORB2 provides an IDL to C++ mapping and is therefore able to communicate with C++ based applications, for example Windows Dynamic Link Libraries (DLLs, see [MICROSOFT01]). The Java ORB provides an IDL to Java mapping and is therefore able to communicate with Java based applications. In order to communicate succesfully over CORBA, the communicating applications should implement the same IDL specification. The IDL specification used for the reference implemetation can be found in Appendix E on page 29.

- *NARAD_CORBA_Client.dll*. This component is the C++ side implementation of the IDL interface. It provides an interface to its clients (e.g. the simulation DLL) using a C++ implementation of the CORBA eventtypes. In other words, the simulation DLL does need to be aware of the fact that the events are to be transmitted via CORBA. The NARAD_CORBA_Client may also be used by other simulation environments than Arena. A specific DLL-component communicating between the NARAD_CORBA_Client and the simulation will then need to be developed.

- *EgasDLL.dll* This DLL functions as a bridge between the C++ oriented service offered by the NARAD_CORBA_Client and the specific interface logic of Arena. The component is able to translate an Arena entity into a standardized C++ event object instance and vice versa.

communicate with other DLLs on behalf of Arena..
- **CorbaDeviceWrapper.** This is the Java based CORBA server, that handles client (e.g. the `NARAD_CORBA_Client`) requests to process events. It basically queues events for further (Java-side) processing, or dequeues events that are to be returned to CORBA clients.

### 4.2.2 Event based communication

The basic communication between the components consists of the exchange of events, in analogy with the rest of the architecture. A first generic event model has been designed in UML and has been mapped into event models usable in the different runtime environments. This event model is meant as a first implementation to test the workings of the communication between simulation and the architecture. It could for example be expanded with a `ControlEvent` subclass for the definition of specific actions to control a simulation (for actions like start / stop / reset). The basic UML scheme is shown in illustration 4.2.2 below, the specification of the specific event schemes per environment can be found in Appendix E. The choice for an object-oriented event model was based on the idea that generalization could be used to hide the details of specific eventtypes. The `Event` class is the superclass (e.g. generalization) of all the other event classes and offers only the most basic functionality needed to route the events to the proper destinations. It was assumed that the exchange of events over the more low-level components (e.g. Narad_Corba_Client, OmniOrb, Java ORB and CorbaDeviceWrapper) could occur independently of the subclass of the

---

31 It also specifies the semantic meaning of the event, by using the 'category' field. 'Notification' or 'Update' are some possible categories.
events. In other words, an instance of PropertyChangeEvent should just be regarded as another instance of Event as far as the CorbaDeviceWrapper, Narad_Corba_Client and ORB’s are concerned. In this way the interfaces of these components would remain the same, whatever sub-classes of Event may later be defined. In practice, however, this turned out to be not completely feasible (see the intermezzo below). Therefore, the interface between Narad_Corba_Client and EGasDLL could not be made completely independent of the eventtype. As a result, this interface will need to be updated as soon as the definition of the eventtypes changes.

The current event model allows for the information exchange of property changes and state changes. The choice to exchange changes of property values rather than, for example, changes of variables was based upon an object-oriented view on the architecture. In this view, each model variable does not exist autonomously, but will always be a property of a certain object. The choice for a state change event is inspired upon Arena’s concept of state changes. Arena allows the definition of a set of potential states of a resource. During the run of a simulation, the state of a resource is always defined as one of the states defined in the resource set. Another reason for the introduction of a state change event was the fact that it allowed the testing of the architecture for different subclasses of events.

**Intermezzo 4: Exchanging Data Between DLLs**

Since DLLs are C++ programs, the information exchange between DLLs basically resembles that of two C++-programs. Basically the following information needs are involved:

- The receiving side should be made aware of the *type* of the event exchanged.
- The receiving side should be made aware of the *data* contained by the transmitted event instance.

There are two mechanisms by which events can be exchanged:

- **By reference.** In this case a pointer (or a 'handle') to the event to be transmitted is exchanged. The receiving side is able to query and update the *original* event using the handle.
- **By value.** In this case a *copy* of the event is transmitted. In this case the receiving side is not able to directly modify the original event.

A final mechanism relevant for this discussion is the concept of *polymorphism*, which means that the exact type of an object instance is determined during runtime, rather than specified during designtime. This mechanism is for example used when an instance of the class PropertyChangeEvent is to perform the role of an instance of Event. Polymorphism (see for example [BROKKEN01] on dynamic casting) in C++ has been defined both for object *values* as well
The C++ polymorphism mechanism, however, acts differently for *references* compared to *values*. This can be illustrated by the following example:

Let a class `A` and a class `SubA`, which is a subclass of `A`, be defined. Let also be defined `SubA_Value`, as an object instance of class `SubA` and `SubA_Reference`, as an object *reference* to an instance of class `SubA`. If `SubA_Value` is now passed as a value to a method expecting an instance of class `A`, the object instance `SubA_Value` will actually be truncated and transformed into an instance of class `A`, which means that the data specific for the class `SubA` will be lost for this instance. Alternatively, if `SubA_Reference` is passed to a method expecting a *reference* to an instance of class `A`, `SubA_Reference` will behave as a reference to an instance of class `A`, while no data will be lost. After the method call, `SubA_Reference` will still be a reference to an instance of class `SubA`.

This example shows that passing of object *values* may result in loss of data. However, communication between different DLLs takes place using object values rather than references. This is because each DLL has its own protected memory space, which causes an object reference to be *illegal* when transferred to another DLL. Therefore, the conclusion is that polymorphism cannot be used in the data exchange process between DLLs.

### 4.2.3 Simulating remote events

The simulation itself should, apart from being able to send and receive events, also be able to provide an intelligible user-interface to *model* the way incoming events should be handled and what outgoing events should be generated. Illustration 4.2.3 shows the way this has been implemented in the lab implementation.

---

32 This distinction is not made in Java, where all variables are treated as object *references*, except for primitive types.
The remote event handling process is an autonomous subsystem in the model, centered around the 'DLL interface' block. The 'DLL interface' block is divided in two subprocesses, a process for incoming events and a process for outgoing events. The remote event handling system does not contain logic to determine whether events should be 'looped back' into the model (compare with Illustration 4.1.7 on page 98). Local changes (that are not to be communicated with the environment) take place via local mechanisms, via direct changes of values instead of an event mechanism. The simulation model therefore resembles more closely the situation of Illustration 4.1.8 on page 99.

The process for outgoing events is quite straightforward: it starts with an access point (an Arena 'station' block), and forwards the entity to the EGasDLL (the 'environment' of the simulation). The Arena entity itself is disposed.

The process for incoming events is slightly more complex. A first difference is that the simulation itself needs to trigger external events to enter the model: external processes should not be allowed to directly access the simulation model during runtime, since this would violate the integrity of the model. The process for incoming events therefore exists of two processes: a process that will periodically check for incoming events and a process that handles events actually entering the model from the environment. After checking event validity (existence of object and property), the events are further processed.

The processing of incoming events can either be 'standard event processing' or more specific

---

33 This is not true for events generated by Visual Basic code, which is triggered internally. Consequently, VB generated events may enter the model directly at the same block as external events would do.
event processing. The standard event processing consists of updating property levels based upon the events received. This kind of processing is needed, since no physical updates of properties are possible for a simulation. Events that should be handled specifically should be redirected at the 'EventLogic' subprocess. This may be the case for events triggering complex control actions.

To model events that should be made available for the environment, special blocks have been designed for the two defined event-types. These blocks are shown in the illustrations below. The general structure of the two blocks is similar: an entity enters the block and is assigned the properties for the appropriate event type. Here also, the change in state or property to be notified is carried through in the model. After that, the entity is duplicated. The original entity will enter a delay block with minimal delay time. This is just to make sure that the other entity, which is assigned to become the remote event, will be processed first. After the remote event assignments, the remote event is sent to the 'process remote event handling' process in the remote event handling subsystem (see Illustration 4.2.3). When the event has been processed here, the simulation will continue. Note that the event handling in the remote event handling subsystem is blocking, which means that the simulation won't continue until the EGasDLL processing has been completed\(^\text{34}\). This sequential processing guarantees that events are broadcast in the right order.

As mentioned before, the event types are designed with an object-oriented environment in mind. The Arena simulation environment, however, is not object-oriented in nature, but flow-oriented. In order to use the object-oriented event types, some design directives have been imposed on simulation design in Arena. These directives are specifically meant to specify the way in which objects should be defined in Arena. In this way at least objects with accompanying properties can be defined in a consistent manner, allowing events to be handled and generated in a generic manner regardless of the objects defined in the

\(^34\) Since the OmniORB2 operations are blocking as well, this implies that the simulation will not continue until after delivery of the event to the CorbaDeviceWrapper.
model. There is no support, however, for advanced object-oriented concepts like inheritance, encapsulation and instantiation. A full description of the imposed design directives can be found in appendix F.

4.2.4 Features not included in the reference implementation

The following technical requirements have not yet been implemented in the reference implementation:

- *Measurement and influence on Quality of Service.* The current implementation relies on the reliable delivery service offered by the TCP/IP protocol. No latency is yet measured or guaranteed.

- *Real-time simulation.* No real-time simulation environment is yet used.

- *Data & time synchronization.* No protocol to exchange data or time information has yet been implemented.
4.3 Conclusions & recommendations

This chapter has presented the research performed on the simulation subsystem. The chapter has therefore first identified functional and technical requirements. On following the lab implementation of the simulation subsystem has been presented.

In this concluding section, these results will be combined, by addressing the research question as introduced in chapter one. After that, recommendation for further research and development are provided, based on the former results.

4.3.1 The research question addressed

This section will address the final research question as posed in the Introduction chapter for the simulation subsystem:

*How should the basic architecture, that makes raw information in a network of distributed devices accessible, be extended with a visualization & control system and simulation capabilities in order to meet its objectives of demonstration and decisionmaking?*

The research question will be addressed by summarizing and combining the requirements and implementations as discussed in the chapter. The requirements identified specify the implicit ‘what’ question present in the research question. The combination with the lab implementation answers the ‘how’ question, e.g. how the requirements are realized in the lab implementation.

The functional requirements for the simulation subsystem, have primarily been identified for the objective of improving decisionmaking support. In addition, technical requirements have been identified as well. This has therefore resulted in 2 concluding tables, that specify the requirements and either indicate where it has been realized in the lab implementation, or indicate that it has not yet been realized. Requirements that have not yet been realized are provided with a comment on the feasibility of additional implementation.
The table below summarizes the requirements identified for the simulation subsystem, in order to support decisionmaking. The functional requirements with regard to decisionmaking have been categorized based on the process model for decisionmaking introduced in §3.1.2, Actor requirements for decision-making on page 36.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evaluation phase</strong></td>
<td></td>
<td>(No requirements have been identified)</td>
</tr>
<tr>
<td><strong>Specification phase</strong></td>
<td></td>
<td>(No requirements have been identified)</td>
</tr>
<tr>
<td><strong>Design phase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>automatic generation of simulation models</td>
<td>no</td>
<td>Several requirements and conditions for the automatic generation and initialization of simulation models have been identified in §4.1.1 on page 84. No fundamental implementation obstacles have been identified.</td>
</tr>
<tr>
<td>simulation model editing</td>
<td>no</td>
<td>Identified as a requirement in §4.1.1, on page 86, but has not been further researched.</td>
</tr>
<tr>
<td>gaming simulation</td>
<td>yes</td>
<td>Multiple actors can interact with the simulation using features of the architectural and the V&amp;C system as introduced in chapter 2.</td>
</tr>
<tr>
<td>distributed simulation</td>
<td>no</td>
<td>Requires a data and time synchronization protocol, see Appendix D for a discussion on one of the protocols available: HLA.</td>
</tr>
<tr>
<td><strong>Implementation phase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soft-commissioning, reality in the loop</td>
<td>yes / no</td>
<td>The lab implementation shows the simulation's ability to represent real devices (needed for soft-commissioning). Reality in the loop requires additional communication between simulation and Virtual Actor, which has no yet been developed (see §4.1.1 on page 89)</td>
</tr>
</tbody>
</table>

*Table 6: Functional requirements identified for decisionmaking*
The table below presents the conclusions with regard to the technical requirements presented in §4.1.2 on page 92.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event delivery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>acknowledged delivery</td>
<td>yes</td>
<td>Realized by the TCP network layer, see §4.1.3 on page 97.</td>
</tr>
<tr>
<td>known latency</td>
<td>no</td>
<td>Requires time-stamping of events + clock synchronization. No implementation obstacles identified.</td>
</tr>
<tr>
<td>event ordering</td>
<td>no</td>
<td>Was not needed for lab implementation: fast network, large inter-event time. Possible mechanisms have been identified in §4.1.2, on page 94.</td>
</tr>
</tbody>
</table>

**Simulation development directives**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>event handling &amp; routing</td>
<td>yes</td>
<td>A central event access &amp; routing model has been implemented in the lab implementation. Exception handling is implemented as well, see illustration 4.2.3 on page 106.</td>
</tr>
<tr>
<td>restricted model access</td>
<td>yes</td>
<td>No direct access to the simulation variables is allowed, communication takes place by means of events only, see §4.2.3 on page 105.</td>
</tr>
<tr>
<td>selective event broadcast</td>
<td>yes</td>
<td>Implemented by dedicated event broadcast components, see §4.2.3 on page 105.</td>
</tr>
</tbody>
</table>

**Simulation execution**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>real-time</td>
<td>no</td>
<td>Realizable by using a real-time simulation environment and Operating System, see §4.1.3 on page 100.</td>
</tr>
<tr>
<td>data / time synchronization</td>
<td>no</td>
<td>Requires a data and time synchronization protocol, see Appendix D for a discussion on one of the protocols available: HLA.</td>
</tr>
<tr>
<td>separate visualization</td>
<td>yes</td>
<td>The standard Visualization &amp; Control system as described in chapter 2 can be used for simulations as well. An additional panel for simulation control needs to be developed, see §3.1.2 on page 43.</td>
</tr>
</tbody>
</table>

**Security**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Realized?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td>Implemented in the basic architecture, see [HISSINKMULLER02].</td>
</tr>
</tbody>
</table>

*Table 7: Technical requirements identified*
It may be concluded that many requirements have already been realized in the lab implementation. For the remaining requirements, some explorations have already been performed. For none of the requirements serious obstacles for implementations have been found. However, in order to meet some requirements, further research & development is required, which will be discussed in the next section.

4.3.2 Recommendations for further research and development

During the chapter, several issues have been identified that require further research and development. This last section recommends and discusses the development of a simulation generation API and a dedicated simulation Visualization & Control panel.

Develop a simulation generation API

In §4.1.1 on page 84, some aspects related to the automatic generation of simulations, given a certain system state, have been presented. It has been concluded that in order to realize automatic generation of simulation as a standard characteristic of the architecture, standard API's should be developed to enable the instantiation and initialization of simulation components. As illustrated in Illustration 2.1.7 on page 21, the functionality of such an API should at least provide the ability to:

- Copy and re-instantiate Virtual Device and Filter components.
- Retrieve simulation components from the Virtual Devices involved.
- Instantiate and initialize the simulation components.

The first two points, may not be difficult to realize, since the Virtual Device and Filter components are highly standardized and well extendible (see [HISSINKMULLER02] and [JACOBS01]). Furthermore, a storage and retrieval protocol for simulation components may well resemble the mechanism already implemented for visualization components (see §3.2.5 on page 67). The realization of the final point (instantiation and initialization), however, is highly simulation package specific. It should therefore be researched whether a generic instantiation and initialization API will be feasible.

Develop a simulation V&C panel

It has been noted in §4.1.2 and §3.1.2 that simulation requires additional visualization and passive control components, like:

- Passive control components for start, stop and reset commands.
- Passive control components for time advancement and speed control.
- Visualization of simulation specific information, like the simulation time.
In addition, a standard protocol needs to be developed to retrieve and initialize this panel.
In the foregoing chapter the integration between a simulation and the architecture was illustrated. Since a simulation or real device cannot communicate directly with a NARAD component, due to differences in runtime environments and logical interfaces, some intermediate components will be required to enable communication between them. The component responsible for this communication is defined as the Wrapper Component, which is therefore a key extension of the architecture, without which no information could be retrieved from any device at all. The Wrapper component’s location in the architecture is shown in Illustration 5.1 and Illustration 5.2. The Wrapper component actually implements a significant part of the functionality defined for the Virtual Device layer in <ref>, in particular:

- **Hiding of technical variety:** The oil company in the EGas case owns over 23,000 gasstations, each consisting at least of 3 devices (a gaspump, Point of Sale terminal and a car-wash installation), which makes at least 69,000 devices in total, dispersed all over the USA. Considering the wide range of vendors and communication protocols involved, it becomes clear that technical variety is indeed a characteristic of the oil company. The wrapper component should hide this variety, by providing a single interface to the Virtual Device.

- **Hiding of technical complexity:** The communication protocols used by devices like gaspumps, are often low-level
protocols. These protocols are often hard to understand (devices may need to be switched by single bits) and delicate (a device may for example receive 50 bits /sec maximum, and shuts down otherwise ). The Wrapper Component should hide this complexity, by providing a high-level interface, that can be used without additional technical restrictions.

These functionalities could have been implemented in the Virtual Device component itself, instead, as presented in Illustration 5.2, in a separate sub-component. However, the sub-component approach has been chosen for the following reasons:

- The sub-component approach eases development, the performance of several wrapper components can easily be evaluated using the same Virtual Device component.
- The approach reflects the different nature of the functionalities realized in the Virtual Device component and the wrapper component. The functionalities realized by the Virtual Device focus on the communication with the architecture (accepting external events, instantiating filters, etc, see [HISSINKMULLER02]), while the functionalities realized by the wrapper components on the other hand focus on communication with the device. One of the differences is, that communication with the architecture is supposed to become highly standardized, while communication with the variety of devices will for the present remain dedicated functionality.

In this chapter some aspects of Device Wrappers in general and the built reference implementation in particular will be illustrated. The chapter starts with the requirements that are to be met by a Device Wrapper. In contrast with the other chapters, this chapter will mainly focus on the technical requirements. This is followed by a discussion of some design choices available. After that, the reference implementation of a device wrapper for devices that can communicate via CORBA is introduced. Finally, some recommendations will be given for future research & development of device wrappers.

### 5.1 Requirements

This section first presents the actor requirements for the wrapper component, which turns out to lead to a need for standardization. After that, technical requirements are identified for the wrapper component.

#### 5.1.1 Actor requirement: low cost wrapper development

In EGas, the oil company owns approximately over 60,000 devices. As has been mentioned in the introduction of this chapter, these devices, with all their differences in hard- and software interfaces, should be wrapped in order to communicate with the architecture. The main requirement from an actor’s point of view, is that the costs of a possible implementation of the architecture may not exceed the potential cost savings it brings. A large part of the deployment costs of the
architecture, however, may well be the development of dedicated wrapper components for the real devices. Although a simple wrapper component has already been implemented during this project, its development is not representative for real-world wrapper development since:

- A real-world wrapper should be developed and tested for minimal downtime. After all, replacing wrappers wrapper components in devices as geographically dispersed as for EGas is expensive. Downtime should therefore minimized as much as possible.
- A real-world wrapper should be developed and tested for extreme situations. The real device wrapped by the wrapper component may be involved in a company’s primary processes. It is therefore unacceptable that a wrapper component may cause the malfunctioning of the underlying real device, regardless whether this happens due to a programming error or overloading. This is especially so for devices posing potential safety risks, like gas tanks or refinery control systems. For such devices all kinds of governmental safety regulations should be met.
- Devices may have ill-documented, technically complicated legacy hard- and software interfaces. To wrap such devices, requires expert knowledge for a considerable amount of time. This again leads to high costs of wrapper development.

These examples illustrate that wrapper development may require significant investments, although additional research will be required for quantitative indications.

However, it is clear, that wrappers need not necessarily be developed per individual device. Many device type communicate using standard hardware interfaces like PLC or RS-232. In addition, many device types may communicate using similar information encoding schemes. To control wrapper development costs, such similarities should be exploited, by defining levels of standardization for different information encoding schemes and runtime environments. In fact, the choice for a wrapper component as such, pluggable in the Virtual Device, with a specific interface (the approach described in this chapter), is already a first attempt for defining a level of standardization for wrapper development.

Although standardization of wrapper components for several platforms and information encoding levels could be useful for legacy devices, manufacturers of new devices may in the future rather want to make complete Virtual Device components available, to have full control on its design and functioning. It can be concluded, that the need for some form of standardization of the device-architecture communication is clear. Further research, among device owners, designers and manufacturers will have to be conducted to determine the most desirable levels of standardization. The next section, which discusses the technical requirements for the wrapper, will in any case introduce some potential standardization levels.
5.1.2 Technical requirements

The main technical function of the Device Wrapper is to receive all the relevant events that are transmitted by the underlying device, and to deliver all the relevant events to the underlying device without disturbing the device, in a reliable and secure way. This function, in combination with the operational context as described in the EGas case raises the need for the following requirements:

- **Low maintenance need.** Device Wrappers may be difficult to service, caused both by the location and number of the Device Wrappers deployed. This is because Device Wrappers run close to the real devices, which may be dispersed over a large geographical area. Since each device should be wrapped by a specific wrapper instance, there may also be a large number of wrappers deployed. Egas’ oil company owns probably more than 50,000 gaspumps in the USA. The need to physically visit these gaspumps to ‘reboot’ the wrapper component in case of malfunctioning, will therefore lead to considerable costs. To control these costs, the maintenance needs (or downtime) of the wrapper components should be minimized.

- **Reusability.** Many types of devices exist, so many types of wrappers may be needed, which raises the desire for reuse of wrapper components. The events are to be communicated with a virtual device, which usually requires a transformation of information into and from the standard Java events (see Illustration 4.2.2 on page 103). Some of these transformations will be device specific, while others will be more generic useful.

- **No significant disturbance** There may be many types of device that will have an external interface that’s not decoupled from the normal device operations. The interface operations may block normal operations or use up the device’s information processing resources. The significance of these effects may differ per device type: a fuel pump that only transmits the data of a completed fuel transaction will probably not be significantly affected if this transmission blocks the device for a tenth of a second. There may also be devices, however, that are required to have high availability and produce many event notifications (like for example a production database system). The operations of such a device would be significantly affected if each event notification would block the device for a tenth of a second. This requirement may be met by heeding attention to the following issues:

  - **Performance** The wrapper components may be designed for maximized efficiency and effectiveness for handling events. This could be specified by measuring the blocking delay experienced by a device during an event notification. Also, if the wrapper components and
the device make use of the same computing resources (as in the case of wrapping a simulation) the average amount of resources used by the wrapper components is a measure of performance.

- **Scalable** The amount of events generated by real devices is not necessarily constant. If the load on a production database increases, the amount of events generated will also increase. The wrapper should in that case be designed in such a way that it is able able to handle this increased load, without using relatively more computing resources. In other words, the operations of the wrapper components should be scalable.

- **Peak load handling** Where scalability has to do with an increasing average load, peak load handling has to do with a relatively large increase of load during a relatively short period of time. During peak load handling, devices may need all available resources to process the peak. This may imply that the wrapper should release computing resources, while on the other hand increased load handling is needed. As a consequence, the wrapper components should have a mechanism to safely dispose of or delay load handling in order to keep the device up and running.

- **Security.** Since the communication between real device and virtual device may take place via some sort of network port protocol (for example socket connections or CORBA), a wrapper may introduce a security vulnerability. Since security is a general requirement of the architecture, this cannot be allowed.

### 5.2 Design choices

There are several design choices available to meet the stated requirements. The design choices that have been encountered during the NARAD project will be illustrated in this paragraph.

#### 5.2.1 Choice of runtime environments

The wrapper components transform native device signals into standard Java events and vice versa. These transformations may be performed by components running in different runtime environments. The choice of runtime environment for wrapper components may be influenced by the following factors:

- **Available functionality.** Different runtime environments offer different functionality. The C++ / Win32 runtime environment, for example, provides support for accessing hardware and memory directly. This feature may be used to directly communicate with a device. The Java runtime environment, on the other hand, does not offer support to access memory addresses directly. It does, however, offer direct access to serial and parallel ports (see [SUN02]).
• **Performance.** Components implemented in different runtime environment may demonstrate differences in performance. The C++ environment, for example, is well known for its efficient code generation and small object sizes. The Java environment does not produce machine executable code, but *bytecode* which is translated into platform specific execution code during runtime (see [SUN03]). The code used to be executed by a platform specific *bytecode interpreter*, which was slow, since each instruction had to be interpreted individually. With the arrival of so-called Just-In-Time (JIT) compilers, the perceived performance difference has become smaller (see for example [SUN03]).

• **Reliability.** A component for a given runtime environment should be written in an accompanying program language for that environment. Programs written in different languages may differ in *understandability* and therefore also in *reliability*. Assembler code, for example, is often perceived to be harder to understand than code of an object-oriented language like Java.

• **Conversions available.** The final runtime environment (of the virtual device component) is Java, so some translation between the different runtime environments should be available. Common conversion protocols that could be used are communication via TCP / IP sockets, communication using CORBA or communication by using the Java Native Interface (JNI, see [SUN04]). It should be noted, however, that the use of an additional conversion, may reduce the performance gains that may have been achieved by using a non-Java runtime environment.

However, the correct weights to be addressed to these requirements is highly device specific, due to the variety in protocols and availability requirements.

### 5.2.2 Data complexity level

The information communicated between the NARAD components, are in the form of Event objects. These objects have a high level of data complexity: they have properties, methods and access levels (private vs. public access). The real devices usually will not be able to receive or transmit data at this level of complexity. Instead, it may communicate using string-value pairs or, also not unlikely, by sending raw bits over some port. The wrapper components should therefore be able to translate to and from a low data complexity level into a high data complexity level.

There are two choices available to implement this conversion. There’s the choice for the *amount* of data complexity levels in the conversion (should a bitstream be directly translated into an object, or is some intermediate data complexity level required?) and a choice for the *components* responsible for the conversion. The consequences of this choice is the subject of the next section.
5.2.3 Monolithic vs. piped approach

In the paragraphs above, it has been shown that choices have to be made with regard to different runtime environments and data translations. Both choices actually imply a choice to develop either a monolithic wrapper versus a wrapper that consists of several subcomponents offering services to each other (see the illustration below). In general, the advantage of a monolithic wrapper can be that it may be optimized for performance, since unnecessary conversions between runtime environments and levels of data complexity may be prevented. The disadvantage of the monolithic approach is, that will not be reusable. No wrapper type will be completely reusable, since each device type ultimately has its specific interface. Some of the components of a piped wrapper, however, (for example a translation from object to key-value pair) may be reusable.

The choices in wrapper design discussed so far are illustrated in Illustration 5.2.2, which shows two wrapper designs for a sample real device, that communicates by means of a proprietary bit-based protocol over a serial port. While the upper example may prove to be a very fast, but not reusable, since internally, a proprietary protocol is used. The lower example consists of more data conversions, thereby probably loosing performance. The CORBA components may be reusable for other devices (or simulations) having a C++ interface. However, in order to realize this reusability, this C++-interface should then be standardized. The only specific component in the lower example will then be the C++-side value converter. Since C++ code is generally considered to be executing faster than Java code, this may also counterbalance the performance loss caused by extra conversions required, compared to the monolithic wrapper.

It may be concluded that the choice for a certain wrapper component design is a non-trivial trade-off between speed and reusability. The speed is non-trivial, since this is determined both by conversions required and by runtime environment performance. The reusability is also non-trivial, since this requires standardization of a component interface, requiring the availability of an authoritative standardization platform.
5.2.4 Local vs. remote wrapper components

Illustration 5.2.2: Two wrapper designs for a sample real device.
Since each wrapper component as well as virtual device and filter components needs to claim computing resources, it may be desirable to distribute the components over multiple computers. This is especially true when the device is *simulated*, since the simulation itself will also claim computing resources. The distribution over several computers will only be possible by introducing some kind of network communication protocol between the components. An example of such a protocol is CORBA (see Illustration 5.2.2), which simultaneously acts as a general bridge between different platforms. A network communication protocol will produce some processing overhead in comparison with a local communication protocol (like inter-DLL communication).

Introducing a network increases the available computing resources, but does so at the expense of the introduction of network latency and limited network capacity. Therefore, running the components on one machine with increased processing resources may be desirable in some situations.

To illustrate the flexibility that is introduced by using a network communication protocol between components, Illustration 5.2.3 shows several distribution policies possible for the same set of components (namely, the set of components shown in Illustration 5.2.2).
However, if a wrapper component is remotely connected to a Virtual Device component, a security leak comes into existence, since a non-authorized person could get access to the wrapper component using the same (RMI) connection. This automatically implies, he/she can access and control the underlying device. Implementing the same security features as implemented for the Virtual Device component (see [HISSINKMULLER02]) might be overdone, especially given the potential high volume event traffic between virtual device and wrapper. An alternative solution is shown in illustration Illustration 5.2.4. The Wrapper Listener and Wrapper Component communicate using TCP/IP addresses B and C, on port 4000. Although the Wrapper Listener is able to contact the host with IP C, traffic arriving from the Internet is not, given that the left machine does not forward incoming internet traffic. The Virtual Device itself is still reachable from the Internet, since the firewall does not block IP adress A. Outside traffic is also unable to directly access the WrapperListener component (which is, in this example, an RMI component listening at port 4000), since all traffic for TCP ports higher than 2000 are blocked by the firewall as well. Finally, the Virtual Device does not even have to be aware of the fact that the wrapper component actually runs remotely, since the Wrapper Listener acts as a transparent bridge.

It can be concluded that the deployment choice for wrapper and virtual device components is related to the nature of the event traffic that will be generated. Again, this is a choice that needs to be addressed per specific device type. When connecting wrapper and virtual devices via a remote connection, security measures should be taken into account.
5.3 Lab implementation

During the project, one wrapper type has been developed, in order to be able to test the feasibility of the wrapper concept and to be able to integrate simulation with the architecture. In this section, an overview will be presented of the current implementation. For a more detailed description of the working of the implementation, the reader is referred to the Java Documentation included on the CD accompanying this report.

In the previous chapter, in particular Illustration 4.2.1 on page 101, the general layout of the wrapper developed has already been introduced. Illustration 5.3.1 presents an additional overview of the wrapper implementation, in the same format as introduced in Illustration 5.2.2 on page 121. This illustration shows that the wrapper has been developed as a piped component design. As a result, several parts of the implementation may easily be replaced.

The components forming the generic wrapper communicate using the Event model described in appendix E. It may be noted that in order to be truly reusable, the end interface (here: the C++ Event model) should be standardized, which at least is related to the collection of devices that may potentially be successfully wrapped via this interface. For this specific wrapper this may for example be the case for an object-oriented simulation environment like Tecnomatix’ Simple++.

Since the Corba and simulation communication aspects have already been treated in the previous chapter, this section will focus on the Java-side wrapper, the Corba DeviceWrapper. A UML overview of this component is shown in Illustration 5.3.2. For clarity, not all methods, properties and packagenames are displayed.
The UML overview shows that the actual CorbaDeviceWrapper extends from the 
_NARAD_CORBA_ServerImplBase class, which is automatically generated, when the IDL
interface (see appendix E, page 29) is compiled by Java's 'idlj.exe' tool. The class implements all
the code needed to interact with Java's built-in Java ORB (see also §4.2 on page 100). In addition,
the CorbaDeviceWrapper implements the IDL's 'NARAD_Corba_Server' interface, which
makes up the Corba side of the CorbaDeviceWrapper's functionality.

In addition to CORBA communication, the CorbaDeviceWrapper needs also to be able to com-
municate with Java components, in particular the Virtual Device components. Since this is true
for any wrapper component, an interface has been defined that may be used by any wrapper com-
ponent, the RealDeviceWrapperInterface. This interface is an extension from the EventSource
interface, a generic interface designed to identify whether objects, whether local or remote, can
broadcast events. The remote event mechanism is further described in [JACOBS01]. The Real-
DeviceWrapperInterface allows virtual devices to (un)subscribe themselves to the wrapper com-
ponent, and allows them to pass events down to the wrapper component.

The final component shown in the example is the 'WrapperQueue' component. It may be recalled
that one of the requirements for the wrapper component is, that the underlying device is not sig-

Illustration 5.3.2: UML overview wrapper implementation
nificantly (which differs per device type) disturbed by the communication between device and architecture. For the Arena simulation, for example, this implied that the event dispatching to Arena needed to be triggered by the Arena simulation. To realize this, queue-objects have been introduced, to control when events are dispatched to Arena. In addition, a queue object has been introduced, to control the event dispatching between the wrapper and Virtual Device. This enables the wrapper to have high availability for events arriving from the device, since dispatching of events to the Virtual Device is implemented as a separate, low-priority thread. As a result of these uncoupling of event dispatching processes, actual queues may of course develop. To prevent that increasing queue-size causes system instability, due to memory shortage, a maximum queue size is assignable, causing an exception to be thrown, when events are dispatched while the queue object has reached its maximum size.

During the test runs performed during the project, this mechanism turned out to be robust enough to take care of the event communication between Arena and the architecture. However, no standard tests are yet available to benchmark the performance and reliability of the developed wrapper component.

5.4 Conclusions and recommendations

This chapter has presented the research performed on the wrapper component. The chapter has therefore first identified functional and technical requirements. On following the lab implementation of the developed CorbaDeviceWrapper has been presented.

In this concluding section, these results will be combined, by addressing the research question as introduced in chapter one. After that, recommendation for further research and development are provided, based on the former results.

5.4.1 The research question addressed

This section will address the final research question as posed in the Introduction chapter for the wrapper subsystem:

_ How should the basic architecture, that makes raw information in a network of distributed devices accessible, be extended with a visualization & control system and simulation capabilities in order to meet its objectives of demonstration and decisionmaking?_

It may be noted that the wrapper component is an essential extension of the basic architecture, that enables the communication between real devices, but also simulations, with the basic architecture. Although the wrapper component could be regarded as part of the basic architecture (e.g. part of the Virtual Device component), the approach in this report has been to isolate its functionality, given its device dedicated nature.
The main functional requirement identified, is the need for some form of standardization of the wrapper component. It has been argued that without such standardization, the development costs for wrapper components, given the large number and wide variety of devices in addition to stringent development requirements, may be considerable.

In addition, the following technical requirements have been identified:

- Low maintenance needs
- Reusability
- No device disturbance
- Security

Finally, a lab implementation is presented, that enables communication between the Arena simulation environment and the architecture. Mechanisms have been identified to ensure security, reliability and reusability. However, it became clear that reusability, although technically feasible, will require a standardization platform, for the definition of data formats and protocols. Although the other requirements were sufficiently met to use the wrapper component as a reliable sub-component during the project, it became clear that measurement standards and tools will be required in order to be able to determine a wrapper component in a useful (e.g. suitable for performance comparison) way.

5.4.2 Recommendations for further research and development

Two recommendations have been identified in this chapter, standardization of wrapper design and development of performance measurement tools & standards.

First of all, research should be conducted to determine the need and possibilities for wrapper component standardization. To determine the need for standardization, the actual variety of devices and corresponding hard- and software protocols should be researched, in order to determine high potential layers for standardization. After that, a standardization platform may be established, possibly by Sun, in order to create a community consisting of device owners, manufacturers and designers, and middleware developers, that provides standard & specific wrapper component and perhaps a marketplace for experts implementing them on-site.

Closely related to such a standardization platform is the development of tools & standards to test the performance of wrapper components on reliability, speed and security. In order to set up these tools & standards, research should be conducted to determine regulations already developed by public and private organizations in the field of device operations. Especially in industries like high-end computer manufacturing (Sun!), chemical productions and space & aircraft manufacturing, such standards are likely to be found.
6 Conclusions & recommendations

This report has presented the research conducted on some extensions of a basic architecture for flexible, real-time monitoring & control of devices. These extensions should leverage the usability of the architecture of supporting actors in their decisionmaking process. The extensions actually researched are:

- A Visualization and Control system
- Simulation
- Wrapper components

For these extensions, the following research question was posed:

*How should the basic architecture, that makes raw information in a network of distributed devices accessible, be extended with a visualization & control system and simulation capabilities in order to meet its objectives of demonstration and decision-making?*

For these three extensions, lab implementations have been developed, using the current technologies available. Although the final implementation has not reached all the decisionmaking functionality presented in the introduction (including automatic generation of simulation models), for each extension a fairly working implementation has been realized.

- A Visualization and Control system has been developed that allows actors to dynamically retrieve visualization components from the architecture, based on powerful, object-oriented queries. The visualization components themselves are implemented as full-featured Java components, which allows the definition of complex visualization components like animation and 3D.

- A simulation environment (Rockwell’s Arena) has been successfully connected with the architecture, delivering a promise that the architecture may once be used to automatically instantiate simulations of devices avialable in the framework.

- A wrapper component has been developed, that is able to take care of the communication between architecture and the simulation package. Using standard Java concepts like threads and queues, a robust wrapper implementation has resulted, that is resistant against fluctuations in supply and demand of architecture and simulation.
Although no fundamental obstacles for further development have been encountered, the lab implementations do not yet meet all the requirements identified for the three extensions. For some requirements not yet realized, the next section will present recommendations for further research and development.

### 6.1 Recommendations

More research and development is required in order to realize the architecture’s full potential to support the actor in his decisionmaking.

#### 6.1.1 Standardization

For all three extensions, issues of standardization were encountered. Since the architecture is developed to be deployable on large scale, standardization is essential. The main issues identified for standardization are the following:

- *The visualization component interface.* This enables the development of a large market for custom, but pluggable visualization and animation components. This may be realized by Sun’s Independent Software Vendors community.

- *The wrapper interface.* Wrapper development costs may be a key component of the architecture’s deployment costs. It should be investigated how many device types and interfaces are out there, which may then be concentrated in a standardization community.

- *Simulation interaction mechanisms.* In order to control simulations remote, via the architecture, a standard interface mechanism should be defined, to prevent simulation environment lock-in.

- *The event model.* The architecture transmits information using events. This event model should at least partially be standardized, to prevent information proliferation.

#### 6.1.2 Development of modeling components

The functionality offered by model components may contribute significantly to the requirement of ‘management by exception’. Further development is therefore recommended.

#### 6.1.3 Development of browsing capability

The current Visualization and Control system is not yet able to provide the actor with a visual overview of the structure of the system he’s interacting with. However, since structural information is already available via the JINI framework, the additional development effort required to enable visual *browsing* is not large.
6.1.4 Real case performance

The lab implementations have not yet been tested in a real-world case. In order to evaluate the validity of the concepts used and the robustness of the implementations developed, it is recommended to set up a small, but extensible, and safe test environment. Egas’ oil company may be able to provide such a test environment.
Bibliography

HISSINKMULLER01: M.W.H. Hissink Muller; P.H.M. Jacobs; N.A. Lang, An introduction on a general framework for distributed visualization & control, 2001


STEI98: Dr. Jeffrey Steinman, Scalable distributed military simulations using the speedes object-oriented simulation framework, 1998


KUHL99: F. Kuhl, R. Weatherly, J. Dahmann, Creating computer simulation systems. An introduction to the High Level Architecture, 1999


JACOBS01: P.H.M. Jacobs, Distributed components in a visualization environment, 2001


LEEUV90: A.C.J. de Leeuw, Organisaties: Management, Analyse, Ontwerp en Verandering, 1990


HISSINKMULLER02: M.W.H. Hissink Muller, A component-based framework for multi-actor decision-support, 2001


SUN03: , About the Java technology, 2001,
http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

SUN02: , Communications API homepage, 2001,


FUJIMOTO00: R.M. Fujimoto, Parallel and distributed simulation systems, 2000
HALSALL95: F. Halsall, Data Communications, Computer Networks and Open Systems, 1995
BOLLELLA00: P. Dibble, S. Furr, M. Turnbull, The Real-time specification for Java, 2000
Auinger99: F. Auinger, M. Vorderwinkel, G. Buchtela, Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop", 1999, in
SHANNON75: R.E. Shannon, Systems Simulation: the art and science, 1975
DIETZ96: J.L.G. Dietz, Introductie tot DEMO, 1996
SPENCE01: R. Spence, Information Visualization, 2001
GORDON89: I. Gordon, Theories of visual perception, 1989


BEER95: S. Beer, Decision and control: The meaning of operational research and management cybernetics, 1966
simulation & distributed visualization

within an architecture for flexible, real-time monitoring & control of devices

APPENDICES
Contents

A Decisionmaking theory

1 Decisions as a mean to solve a problem......................................................iv

2 Requirements for an information architecture..........................................v

B The topic implementation

1 The Topic elements......................................................................................viii

2 The UML overview......................................................................................x

3 Value comparisons......................................................................................xii
   3.1 The Compare type comparison...............................................................xii
   3.2 The standard equality test......................................................................xii
   3.3 No value comparison.............................................................................xiii
   3.4 User defined equality test......................................................................xiii

C The dynamic object editor

D Introduction on the High Level Architecture

1 Introduction..................................................................................................xvii

2 Architectural overview................................................................................xvii
   2.1 The Object Model template....................................................................xix
   2.2 The HLA Interface definition..................................................................xx
2.2.1 Federation Management ........................................... xx
2.2.2 Declaration Management ...................................... xxi
2.2.3 Object Management ............................................. xxii
2.2.4 Ownership Management ....................................... xxii
2.2.5 Time Management ............................................. xxiii
2.2.6 Data Distribution Management ............................. xxiv
2.2.7 Support services ................................................ xxv
2.3 HLA Rules .............................................................. xxv

3 HLA made practical...................................................... xxvi
  3.1 A typical implementation ......................................... xxvi
  3.2 Support tools available ........................................... xxvii

4 Conclusion .................................................................... xxvii
  E Eventtypes & interfaces
   between simulation and Java

1 Sequence diagrams ................................................... xxviii

2 IDL interface specification ........................................ xxix

3 Event model & translations ....................................... xxxii
  F Imposed requirements for the
   distributed simulation model
List of figures

Illustration 6.1.1: Simple control system.................................................................iv
Illustration 6.1.2: Information flows...........................................................................v
Illustration 6.1.3: Equal Marshalled Objects..............................................................vii
Illustration 6.1.4: The Matching process illustrated....................................................viii
Illustration 6.1.5: The 'Match' process illustrated.......................................................x
Illustration 6.1.6: UML diagram of the Topic classes..................................................xi
Illustration 6.1.7: The dynamic object editor in action................................................xiv
Illustration 6.1.8: point property introspection............................................................xv
Illustration 6.1.9: The collection editor.......................................................................xv
Illustration 6.1.10: A federation overview.................................................................xviii
Illustration 6.1.11: Example interaction model...........................................................xix
Illustration 6.1.12: Example object model.................................................................xix
Illustration 6.1.13: Service categories defined for the HLA interface...........................xxi
Illustration 6.1.14: A region in a namespace spanned by 3 dimensions........................xxiv
Illustration 6.1.15: Sample RTI implementation..........................................................xxvi
Illustration 6.1.16: Sequence diagram of event broadcast...........................................xxviii
Illustration 6.1.17: Sequence diagram of poll for external events.................................xxix
Illustration 6.1.18: Eventtypes used in the architecture.................................................xxxii
Illustration 6.1.19: Screenshot of variables in a wrapped Arena simulation...................xxxiii
Sun’s customers regard their information systems ever more as a decisionmaking support tool in addition to the traditional information database. This section therefore discusses some theoretical notions on decisionmaking and their implications for an information infrastructure. A model of decisionmaking is introduced that focuses on decisionmaking as a (continuous) process driven by a certain problem perception. The section concludes with some distilled requirements for an information system providing decisionmaking support.

1 Decisions as a mean to solve a problem

In [BOTS97] decisions are thought to emerge out of an underlying problem perception of an actor, the problem owner. It therefore introduces an analytical model explaining the relationship between a single, rational actor, a problem that’s perceived by this actor and the relationship this actor has with regard to the observable system in which the problem occurs. A graphical view of this model is presented in Illustration 2.1.2. This model assumes that the world can be regarded as a system, a collection of elements and relations surrounded by an environment. It also assumes that the actor is the only actor involved in the decision. The model illustrates that the problem owner should:

- Be aware of his problem perception. The problem perception may for example be formalized by
means of an objective hierarchy or means-ends model.

- Observe a specific part of the world, which is bounded by and related to the problem perception. This part of the world will be identified as the relevant system. The whole system that can be observed by the actor will be identified as the observable system.

- Be aware of the way instruments and environment influence the relevant system. This implies that a model of the relevant system is available for the problem owner to enable him to predict in advance the consequences of a certain course of action for a given environment behaviour.

- Have enough instruments to act on the system. The problem owner should at least have enough instruments at his disposal to counterbalance the environmental impacts on the relevant system.

The aforementioned consequences correspond with the five necessary conditions for effective control:

- Clear objectives of the control process.

- Availability of a model of the controlled system.

- Available information on the state and environment of the controlled system.

- Availability of enough steering instruments.

- Availability of sufficient information and communication handling capacity.

The last condition follows from the need to process state changes of the relevant system fast enough in order to be able to apply suitable control mechanisms in time.

## Requirements for an information architecture

The illustrated decisionmaking model implies the following consequences for an information architecture providing decision support:

- The actor should be able to put boundaries on the observable system in order to limit the amount of information received by the actor. The need for such a limitation of the received information is shown in Illustration 6.1.2. The actor is only interested in the information in the relevant system and therefore needs to filter the superfluous information. The theory of systems suggests several dimensions for bounding systems, under which the following:

a) Definition of a subsystem, consisting of a subcollection of elements of the original system (including the attached relations). The elements of the subsystem are usually characterized by a certain condition. All elements may for example be located in a specific geographical area.

35 See [LEEuw90].
b) Definition of an aspect system, consisting of a subcollection of *relations* of the original system (including the attached elements). The relations of the aspectsystem are usually characterized by a certain condition. All relations having to do with information flows may for example be included.

c) Definition of a phase system, in which a system is observed during a certain *time-interval*\(^6\). The chosen time-interval is usually defined by a certain condition, for example maximum load on a system.

- The actor should be able to *monitor* the state of the system and its environment and be *notified* of changes, in order to be able to react accordingly, given the problem perception.
- The system information should be *visualizable* in such a way that it gives the actor the most insight in the relevance (and significance) of the information for the problem perception. In this way, the information and communication handling capacity of the *actor* may be leveraged. Since problem perceptions as well as information types differ, various ways of information visualization should be available for the actor.
- The actor should be able to use the information available of the observable system as a basis for a descriptive or predictive *model* of the system. If this model is to be a computer model, this implies that information is available in a general computer-readable form.

The actor should be able to act on the system by use of his instruments. The information system could support this for some instruments by sending *control* information to parts of the relevant system.

---

\(^{(6)}\) Once or periodic
The topic implementation is a Java-based, object-oriented query mechanism. It has been designed especially to provide a semantic powerful query mechanism for objects, without giving up object-oriented features like encapsulation and inheritance. In Appendix D of [JACOBS01], the topic implementation is also compared with the JavaSpace / Entry based query mechanisms. The intermezzo below summarizes his conclusions.

This appendix will introduce the topic implementation by focussing first on elements and relations involved in the topic querying process. After that, a formal UML diagram will be presented, which will finally allow for the presentation of some specific details of the topic implementation. The Topic implementation may be found on the CD accompanying this report, in the package com.sun.narad.topic.

**Intermezzo 5: Current Java Object Query Mechanisms**

The *JavaSpace* technology compares objects by using Marshalled objects. Documentation on Marshalled objects can be found at [SUN08]. Basically, Marshalled Objects contain a *serialized* object (see [GOSLING00]), with *codebase* information added. Using this codebase information, an object receiving a Marshalled Object is able to retrieve the class information for the serialized object contained in the Marshalled Object. This is explained in Sun’s documentation on Remote Method Invocation (RMI), which is also available at [SUN08].

JavaSpaces are one of the mechanisms used in Jini to compare *entries*. The entry concept is a Jini-based
concept (see [JACOBS01] for a detailed introduction on Jini), that is used to identify Jini services.

The mechanism used by a JavaSpace to compare entries, is to marshall the entries and compare them bitwise. A comparison in this way is useful, since the Marshall mechanism guarantees that objects of the same class with the same property values, will be serialized in the same way. This mechanism is illustrated in Illustration 6.1.3. Since only the bitstreams have to be compared, the comparing object does not need to be aware of the classes of the original objects. In addition, it is also a fast method.

There are also two major drawbacks of this mechanism:

- Any comparison only takes the ‘equal’ relationship into account. In other words, it’s only possible to test for equivalence of the serialized object stream. It is therefore not possible to test whether the value of property A of object 1 does not differ less than a certain value: only equality can be tested.

- Little support for ‘special’ comparison. Usually, when comparing to composite objects, only one or two properties need to be compared. The JavaSpace technology is only able to support this, when these properties have public access. This, however, is a direct violation of the object orientation principle of encapsulation, which tells us that properties should only be changeable by their corresponding object. In other words: an object should have private properties.

It have been these two drawbacks that have amplified the need to develop a new object-oriented query mechanism.

### 1. The Topic elements

The illustration below presents the elements involved in using a topic object. The basic use of the topic object, once it has been constructed, is to test whether another object (e.g. a specific instance of some object class) meets the requirements for the topic, in other words, falls under it. The topic primarily decides on this by comparing the specific instance with its internal template object. The template object has been added by some user to the topic, in order to specify an example object that defines the object collection that fall under it. Any serializable object instance may be used to form the template, so the object orientation paradigms are not violated.

The example shows a template object of the class 'Vehicle'. This Class contains several properties, that

Illustration 6.1.4: The Matching process illustrated
are illustrated by a property graph. Since properties may themselves be objects again, such a graph may contain several levels. Each node in the graph potentially has a corresponding node in the specific object instance. The basic idea of the topic comparison process is that potentially each node of the specific instance:

- Should be an instance of the node type of the template.
- Should meet the specific value requirements for that node.

The specific value requirements are stored separately in the Topic object in the Modifier object. This object allows the definition of a certain Operator for a certain property. The example shows that the Operator '<' has been defined for the node corresponding to the property 'vehicle.engine.power'.

While in theory the comparison criteria should be applied to all the nodes in the template object, in practice a top-down process is executed, in order to limit the processing time needed. This process will now be illustrated by continuing the example.

The matching process is top-down. This implies that the first node that will be compared, is the upper one. Execution of the comparison steps for that node results in the following:

- Given that the 'Car' class is a subclass of the 'Vehicle' class, it follows that this specific instance is indeed an instance of 'Vehicle'. Typecheck is thus done in an object-oriented way, since inheritance is explicitly supported.
- There is a Modifier object available for this node. Its operator is 'any', which means that on this node level, its value is not important.

After this first step, the property graph of the template object is explored one level deeper, to check whether there are any more properties, and whether Modifier objects have been defined for these properties. Identification of the next level of properties is performed using the 'java.beans' package, by calling the 'Introspector.getBeanInfo()' method (see [SUN09] for details on the JavaBean architecture). The returned BeanInfo object contains all the details on the classes' properties. The next property for which a Modifier object has been defined in the example is the 'engine' property. The other graph nodes, for which no Modifier objects have been defined, are not further explored. Since the Modifier operator of the 'engine' property is again 'any', the process will start to deepen a final level. On this final level, two properties, namely 'type' and 'power' are available.

The corresponding operators now are not 'any', but '=' and '<'. Now, actually some property values will have to be retrieved from template and specific object instance, in order to decide whether they meet the criteria. The BeanInfo object provides the functionality to retrieve property values by property name. It does this by searching the class for a 'get<property>' method and invoking this method on the source object. In this way, the properties are retrieved by calling a method, instead of direct introspection. The object's properties therefore do not need to be public. The final step in the example is to compare the retrieved property values and decide whether the operators are satisfied. Since this is the case in the example ('1.3 = '1.3' and '50' < '60'), the match will return 'true'.
The process described informally in the example may be described formally by the process in Illustration 6.1.5. Note that the process 'match subtemplate' is recursive; the match process is called again, but with a different object, namely one level deeper in the object graph. The final stop condition for the match method is the fact that there are no more properties with a Modifier left in the object graph, while all the submatches have returned 'true'.

*Illustration 6.1.5: The 'Match' process illustrated*
The UML overview

Illustration 6.1.6 shows a UML diagram of the main classes of the 'com.sun.narad.topic' package. Based on this diagram, some implementation details will be discussed. More details can be found in the JavaDoc generated documentation on the CD supplied with this report.

- **TopicInterface**: Besides the 'match(Object)' method, the interface also provides methods to get and set the operators for the template. The parameter of these methods is a String denoting the property's place in the object graph (for example "car.engine.type").

- **Modifier**: This object is the datastructure used to link Operators to properties. The modifier object shown in Illustration 6.1.6 actually consists of a couple of modifier objects, that are related to each other by the parent and child relations respectively. Although a parent relation would suffice to define the datastructure, the 'child' relation has been added to ease the top-down search process.

- **Operator**: This class contains constants that define allowed operators (like '=' , '<', etc.). The 'checkOperatorCompatibility(Class, operator)' method is used to check Class compatibility for a given operator. This method will for example return 'false' when a non-primitive Class is used that does not implement the 'Comparable' interface, in combination with a '>' operator. This is because in this case it is not clear in which way the property values should be compared. It must be noted that the 'java.lang.Comparable' interface is the standard Java interface used for comparisson. It basically contains one method with signature 'public int compare(Object)'. The integer returned may be negative, positive or zero. Zero is defined to denote equality.

The 'operate(Operator, int)' method translates an outcome of the 'compare' method (e.g. the supplied integer) into a boolean, based on the Operator supplied. When comparison of a & b results in a
negative number (meaning that \( a < b \)) and the operator’s value is ‘\(<\)’, the operate method will return ‘true’.

- **Topic**: It may be noted that the Topic’s template is supplied as the constructor’s parameter.

### 3 Value comparisons

This final section introduces the way in which property value evaluations are performed. It will be shown that this differs for property classes as well as operators.

The Operator class defines several standard operators that may be defined in a topic. The following list provides these operators, categorized by comparison process:

- **Compare type comparison**: GREATER ("\(>\)"), GREATER_EQUAL ("\(\geq\)"), SMALLER ("\(<\)”) and SMALLER_EQUAL ("\(\leq\)").
- **Standard equality test**: EQUAL (\(=\)) and NOT_EQUAL ("\(!=\)").
- **No value comparison**: ANY ("ANY").
- **User defined equality test**: USER_EQUAL ("U==") and USER_NOT_EQUAL, ("U!=").

These operations need to be applicable to different kinds of values, in particular to:

- Primitive types (int, char, byte, etc.)
- Objects

The identified processes will be discussed in the remaining subparagraphs.

### 3.1 The Compare type comparison

This mechanism is used when the class of the currently checked property implements the java.lang.Comparable interface. This interface is the standard interface for determining the order of objects. This mechanism is used to compare values of primitive types. This is possible because:

- The BeanInfo object will return the primitive Object wrapper class for a property having a primitive type. This means that for a property of primitive type 'int’, the Integer class (its wrapper class) will be returned. This works in both ways: when setting such a property, again an instance of a wrapper class should be used. The translation into a primitive type is therefore part of the Bean implementation.

- The object wrapper classes for the primitive types (classes like Integer, Double) do implement the Comparable interface. One exception (the Boolean class does not implement Comparable) is therefore not handled correctly by the topic in combination with Operators like SMALLER. The topic should not accept this operator for the boolean type.

As mentioned before, the Comparable interface contains one method ‘compare’ that returns an integer
value. Since this method returns '0' when the values compared are equal (exceptions on this rule are rare: the topic provides the 'user-equal' method to deal with this exception), this mechanism may also be used to test for equality.

### 3.2 The standard equality test

The standard equality test extends the mechanism just described. This means that when 'java.lang.Comparable' is implemented, the 'Compare type' comparison mechanism will be used to test for equality. If not implemented, the standard equality test is performed, which is based upon the characteristic of equality of Marshalled objects having the same properties (see page 7 of this appendix). This means that both values are Marshalled, after which they are bitwise compared.

### 3.3 No value comparison

This process is performed in case of the 'any' operator. Basically, no value comparison is performed at all, the comparison will therefore always return 'true'.

### 3.4 User defined equality test

The final option is to use a user-defined equality test, instead of the standard Marshalling process just described. Java again provides a standard mechanism for this purpose, since Java's root class (java.lang.Object) contains the 'equals' method (signature: public boolean equals(Object)). Since each object in Java extends java.lang.Object, each object contains this method. A class may have overridden the standard method implementation, which enable users to define their own implementation.

A user can specify to compare values using the 'equals' method instead of marshalling, by specifying USER_EQUAL or USER_NOT_EQUAL as the operators.
The dynamic object editor introspects a JavaBean class or instance, in order to automatically generate a GUI to edit its properties. An implementation of such a dynamic object editor has been built and can be found in the 'com.sun.narad.util.beans' package. This appendix will describe the workings of the editor by first introducing the introspection mechanism. After that, the way an object’s composition graph is processed will be presented.

The introspection mechanism is facilitated by the standard JavaBean property introspection object, the 'java.beans.Introspector'. This introspection mechanism determines the properties of a class or instance by checking an object’s methods for the property pattern: get- and setmethods. The mechanisms provides the possibility to retrieve and set these values by use of these methods. Object attributes therefore need not to be publicly exposed. In addition, it is possible to limit or extend the properties thus exposed by defining BeanInfo\(^{37}\) classes. Details on the introspection mechanisms and the BeanInfo object may be found in the standard Java API, see [SUN08].

Illustration 6.1.7: The dynamic object editor in action

Being able to introspect the top-level properties of an object, does not provide the complete functionality needed: many properties are themselves again objects, and thus potentially subject to introspection. Illustration 6.1.7 shows some the range of property types that are to be resolved by the editor. The illustration shows an object graph with five properties: point, color, listener, collection and array. Besides their names the types and values before introspection are shown. The right part of the illustration shows the output of the dynamic object generator when such an object is processed. The different properties

---

37 These will also be transferred when beans are retrieved from a remote location, using RMI.
will be processed by the dynamic object editor as follows:

- The *point* property is itself an object, containing for example X and Y axis coordinates. For this property, therefore a default object editor is created. The 'editor...' button will trigger further introspection of this property. The result is shown in Illustration 6.1.8. It may be noted that an instance of Point is automatically generated.

- The *color* property is of type java.awt.Color. In the Java 2 runtime environment, a standard customization panel is *registered* for this kind of objects. Therefore, the object editor is able to retrieve a standard color property customizer, including a color chooser component.

- The *listener* property is an interface, not a class. When this property would not have a standard value, the introspection mechanism would fail, since it cannot decide which class implementing the interface should be instantiated. In this case, however, the property's standard value is an instance of the class MyButton. Therefore, a default object editor for this instance of MyButton is generated.

- The *collection* property is of class java.util.Vector, which implements the java.util.Collection interface. This interface has become the standard to be implemented by any object that implements a multi-object data structure. Ideally, the dynamic object editor would recognize the that the Collection editor is implemented and generate a special GUI that enables modification, deletion and addition of the Collection members. In the current version, this is not yet supported. Instead, a default object editor is generated.

- The final property is the *array* property. The array type is the 'old-fashioned' way to represent a (structured) data collection. The current implementation is able to introspect a one-dimensional array (of any type) successfully. A collection editor will be generated to provide GUI support for
array property editing, allowing object addition, modification and deletion, which is presented in Illustration 6.1.9.

It has been shown that most of the possible scenarios are handled correctly by the dynamic bean editor. A future version should, however, also provide support for objects implementing the java.util.Collection interface, by offering the same kind of support as is already offered for one dimensional arrays.
Introduction on the High Level Architecture

1 Introduction

The High Level Architecture (HLA) is an architecture that provides the possibility to combine several computer simulations, running on different machines, into one large simulation. It was developed during the cost-conscious 90’s, when the US Department of Defense (DoD), a heavy user of simulations, had a clear need to leverage the benefits of simulation investments. The HLA was to fulfill this need by increasing the reusability and flexibility of DoD’s many simulation models. According to [KUHL99], the Defense Modeling and Simulation Office (DMSO) started the development of the HLA in 1995. It soon became clear to them, that an HLA-definition without broad acceptance within and beyond the DoD would prevent a structural accomplishment of the objective. Therefore, the development process for the HLA was carried out in several stages, that involved a diversity of teams from industry, government and laboratories. This approach resulted among others in the following deliverables:

- March 1996: Delivery of an initial technical architecture.
- August 1996: Delivery of the baseline HLA 1.0 definition, approved as a standard technical architecture for all DoD simulations in September 1996.
- 1998: Adoption of HLA technology by the Object Management Group
- September 2000: Adoption of HLA as an official IEEE standard (1516, 1516.1 & 1516.2)

Besides the specification processes, also a support tools development process was started. The resulting tools are available online, at the DMSO software distribution center (http://sdc.dmso.mil/).

In the remainder of this paper, the parts and workings of the HLA will be discussed. The next chapter will give a broad overview of the different aspects of the HLA-definition. The third chapter will discuss some practical HLA-implementation available today. Finally, the paper will be concluded by a short conclusion on the usability of HLA for distributed real-time simulation systems.

2 Architectural overview

As mentioned in the introduction, the main purpose of the HLA is to provide a architecture for connecting several simulations into one large simulation system. The HLA does just that: it defines an architecture, but does not define its implementation. It is useful to first give an overview of the different definitions that make up the HLA. After that, the different parts of the HLA specification will be discussed.
The following overview of definitions within the HLA is based on [KUHL99]:

<table>
<thead>
<tr>
<th>Notion</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Federation</td>
<td>The combined simulation system created from the constituent simulation</td>
</tr>
<tr>
<td>Federate</td>
<td>Each simulation that’s included in the federation</td>
</tr>
<tr>
<td>Federation execution</td>
<td>A session in which the federation execute</td>
</tr>
<tr>
<td>Runtime Infrastructure (RTI)</td>
<td>Supporting software in the federation: to be distinguished from the federates</td>
</tr>
<tr>
<td>Object Model Template(OMT)</td>
<td>The HLA meta-model for all the object models defined for a federation</td>
</tr>
<tr>
<td>Federation Object Model (FOM)</td>
<td>The object model describing all the objects that may be shared within the federation</td>
</tr>
<tr>
<td>Simulation Object Model (SOM)</td>
<td>The object model describing all the objects that occur in a simulation (e.g. a potential federate)</td>
</tr>
</tbody>
</table>

**Table 8 Some HLA notions defined**

These definitions make up for a first perspective on the HLA, depicted in Illustration 6.1.10: A federation overview, on page 18. The following things may be observed:

- There’s only one RTI in one federation. However, this RTI may itself be implemented in a distributed way.
- The federates always interact with each other through the RTI. This rule is meant to enforce the reusability of the simulations.
- The notion ‘simulation’ with respect to HLA is a broad notion: it may be a complete simulation, but also a passive data collector / processor, like a visualization panel. In the extreme, it can be any component (wrapped or not), as long as it’s able to communicate via the HLA interface with the
RTI.  

- Officially, each federate is fully described by its SOM, while the federation is fully described by its FOM. However the RTI needs to be aware of the objects that are to be shared within a federation. Therefore, a federation will be able to execute without definition of the SOMs, but won’t be without a defined FOM.

The aforementioned definitions are fully described in the HLA specification. The HLA specification in its current form consists of the following parts:

- The Object Model Template ([DOD98_OMT]).
- The HLA interface definition ([DOD98_I] and [DOD98_API]).
- The HLA-rules, which define ‘good practices’ which should be obeyed in order to be HLA-compliant ([DOD98_R]).

In the following paragraphs, each part of the specification will be discussed shortly, in order to refine the initial overview.

### 2.1 The Object Model template

The Object Model Template (OMT) is the meta-model that underlies all the object-models that are defined for the HLA, specifically the SOMs and the FOM.

An object model based on the OMT consists only of object classes and inheritance relations. Object classes may have any number of attribute, however, these are instance attributes, the OMT does not support the notion of static members as found in languages like C++ or Java. Furthermore, OMT objects don’t have behaviour. Besides this object metamodel, the OMT also specifies procedures and documentation rules for object models. These shall not be discussed here, they can be found in [DOD98_OMT].

Examples of OMT-based object models are shown in the illustrations below.

In both examples there’s one root-object, with several objects inheriting thereof. The naming convention to denote a certain object class is as a fully qualified tree, with dots between the nodes. The manager class in illustration 6.1.12 for example would be denoted by 'ObjectRoot.Manager'. The HLA is not...
case-sensitive for classnames. However, during federation execution, objects and attributes will most of the time be identified by *handles*, which are integers that the RTI assigns for all objects and attributes in the FOM.

The examples shown here are actually part of the FOM-definition. The FOM consists of two separate object models, the Object Class Hierarchy (OCH) and the Interaction Class Hierarchy (ICH). The OCH is used to define objects with a certain state, that are to be shared within the federation. The ICH is used to define messages or events that are to be sent within the federation: they don’t correspond to objects with state.

Each FOM consists of standard Object and Interaction Hierarchies, which are extended by user-defined objects and interactions. The standard hierarchies are also known as the Management Object Model (MOM). The MOM enables federates to receive management information from the RTI during federation execution. The examples shown above actually show a part of this MOM definition. The complete MOM, however, consists of over 40 classes. The full MOM-specification can be found in [DOD98_I].

### 2.2 The HLA Interface definition

The HLA interface definition is the core of the HLA specification. The complete interface definition consists of two parts:

1. *The HLA-interface document ([DOD98_I]).*
   
   This document describes the different services that should be available for the RTI and the federates in order to be HLA compliant. The services are described in an abstract way, without references to implementations.

2. *The HLA-API-interface document ([DOD98_API]).*
   
   This document provides reference API’s for an HLA-interface, currently for IDL (CORBA), Java, C++ and Ada 95. These API’s enable developers to actually build HLA-compliant simulations and RTI’s.

The HLA-interface specification defines several categories of services, each with their own mechanisms and application area. These are shown in Illustration 6.1.13: Service categories defined for the HLA interface. The illustration also shows that the HLA interface is a true interface: it needs to be implemented by federates and RTI. Although not specified by the interface specification, it is possible to add services to the HLA-interface. This is shown in the illustration by the ‘non-HLA service definition’ category. In the following paragraphs, the different service categories will be introduced.

#### 2.2.1 Federation Management

The services within the federation management category are used to actually start a federation. The following services are included:

- *Start a federation.*

  One federate may request to start a federation, by providing the correct FOM to the RTI.
- **Register as a federate.**
  After start-up of a federation, federates may join the federation. Since a RTI can execute multiple federations, a federation has to be selected by a federate. Federate registration may take place at any moment during federation execution. It’s the federate’s responsibility, though, to request updates of objects it needs for initialization.

- **Resign from a federation**
  A federate may resign from a federation at any time by requesting this service. However, a federate is responsible for a ‘clean exit’. It should for example transfer or delete shared objects that is was responsible for during federation execution.

- **Destroy a federation**
  After execution of a federation, a federate may request the end of the federation. Each federate gets the chance to make a ‘clean exit’ before the federation is actually ended.

### 2.2.2 Declaration Management

Declaration management services are used by federates to announce subscription or publication of federation data. The RTI acts as the register where publications / subscriptions are managed. The mechanism works as follows: at any time a federate may notify the RTI which FOM-data it wishes to subscribe to and which FOM-data it intends to publish. So do all the other federates.
Each federate will send data it produces to the RTI. The RTI will then lookup registered subscribers and deliver the published data to them. The publishing federate is not aware of the number of subscriptions\textsuperscript{38}. This enables federates to register or resign from a federation during execution without problems.

2.2.3 Object Management

In any federation there are just two type of data a federate can subscribe to: objects & interactions. As mentioned before, the difference between them is the fact that objects have state and do persist a significant period of time, while interactions behave like events or single messages: they only exist during their transmission and rather represent state changes. For both modes the RTI provides the option to deliver data reliable or unreliable.

The interaction mechanism is straightforward: a federate may begin sending interactions after having announced its intent to publication to the RTI. The object mechanism is more elaborate. First of all, a federate needs not to publish all the attributes of an objectclass\textsuperscript{39}. Secondly, a federate needs to register an instance of an object class before it can update its values. The RTI will notify subscribed federates of the registration, after which the publishing federate may begin to update the instance’s attribute values. Finally the publishing federate may announce deletion of a registered object.

2.2.4 Ownership Management

An object attribute may at a certain moment in time only be updated by one federate, that federate is said to own the attribute. However, shared objects usually enact in processes that involve several federates, therefore, it would be favourable if the ownership could be transferred to other federates. This is exactly what the ownership management services provide.

The following modes of transferring ownership are provided:

1. Divestiture:
   a) Unconditional ownership divestiture.
      In this mode the owning federate immediately divests ownership. As a result, the divested attribute(s) will have no owner and will not be updatable before a federate has acquired their ownership again.
   b) Conditional ownership divestiture.
      In this mode, the owning federate publishes its intent to divest the ownership of an attribute. If another federate then requests ownership of the attribute, the ownership will be transferred. If no federate is interested, the owning federate may withdraws its intent to divestiture ownership\textsuperscript{38}.

2. Acquisition
   a) Active ownership transition.
      In this mode, the federate that wants to acquire ownership requests transition of ownership to the RTI. The RTI will relay this request to the owning federate. The owning federate responds with

\textsuperscript{38} With one exception: the RTI can notify the publishing federate when no subscriptions are registered.

\textsuperscript{39} An interaction, in contrast, will always be sent with all parameters.
an approval or denial. Again, the requesting federate may withdraw its request\textsuperscript{40}.

\textbf{b) Passive ownership transition.}

In this mode the federate requests ownership acquisition to the RTI. The RTI will check whether the attribute is currently owned. If not so, the ownership will be transferred to the requesting federate, otherwise, an exception will be sent.

\textbf{2.2.5 Time Management}

Time management services are used to preserve the logical arrival sequence of events within the federation and to control the federation execution speed.

Events sent within the federation can be \textit{receive order} (RO) or \textit{time-stamp order} (TSO). RO events will be delivered by the RTI in the sequence the RTI received them. TSO events have a timestamp. The RTI will deliver them to a subscribed federate based on the internal virtual time of that federate.

The HLA offers several modes of time services to the federates, however, each federate is free to choose the time service it needs. In general, a federate can be:

\begin{itemize}
  \item \textit{Unregulated}. The federate receives and sends all events in \textit{receive order}.
  \item \textit{Strictly time constrained}. The federate manages internal virtual time and may receive TSO-events. Events will be sent as RO. Advancements of internal virtual time are requested to the RTI. The RTI will only grant advancements when the requested time is earlier than the earliest internal time of any time regulating federate (see below).
  \item \textit{Strictly time regulating}. The federate manages internal virtual time and does not depend on other federates for time advancement. However, since time constrained federates need to be aware of its time updates, the regulating federate notifies the RTI of any time advancement. The time regulating federate will receive any event in receive order and may send TSO-events.
  \item \textit{Time constrained & regulating}. The federate’s virtual time regulates the federation, but is also dependent of the federation global virtual time. Time constrained & regulating federates may send and receive TSO-events. To prevent lock-up situations, the federates will usually (but not necessarily) use a look-ahead value.
\end{itemize}

Within these modes, the HLA supports several time advancement schemes, notably:

\begin{itemize}
  \item \textit{Time vs. event based time management}. A federate may request time advances in a time based manner, e.g. by advancing certain periods of time. While the time advance request has not been granted, the RTI will collect events for the federate until the time can be granted: the RTI will then deliver the events and grant the request.

  It is also possible for a federate to advance time in an event based manner. The federate then requests time advancement to its next internal event time. If no federation events occur, the RTI will grant the request. Otherwise, the request will be granted for the time the next federation event

\textsuperscript{40} It’s the RTI's responsibility to solve timing-problems that might occur due to network latencies. It does so by serializing incoming messages in one logical queue and sending exceptions for messages that have become out of context.
arrives.

- **Pessimistic time management.** In this scheme the RTI guarantees that no TSO-events will be delivered earlier than the current federate's virtual time. However, this scheme also prohibits 'fast' federates to run ahead of the federation.

- Optimistic time management. In this scheme, the RTI does not guarantee that no events arrive in a federate's past. However, each event will be accompanied by a retraction handle, that permits a federate to roll back an event and therefore restore the federation to a logical correct state. More information on pessimistic and optimistic time management can be found in [STEI98].

### 2.2.6 Data Distribution Management

As mentioned in the description of declaration management, the main mechanism for data exchange in HLA is via publication & subscription. However, this mechanism might yield too much overhead in certain situations. For example, when simulating radio traffic messages, only receivers within a certain range and on the same frequency need to receive a radio transmission. Data Distribution Management enables just that by enabling publishing and subscribed federates to bind with a certain **region** in a certain **namespace**.

The mechanism works as follows:

- In the FED, a **namespace** should be defined. A namespace is spanned up by a number of **dimensions**.

- Object attributes or interactions that are to be published in a certain namespace should be associated
in the FED with the namespace.

- During federation execution, federates can then create and modify regions in a namespace. A region is build up from different extents. An extent is a limited space within the namespace, which has a lower- and upper bound for every dimension. A region is the collection of the extents defined in the namespace. Illustration 6.1.11 on page 19, shows a region, consisting of one extent in a 3-dimensional namespace.

- During publication the RTI will check whether the regions of the publisher and subscribers do overlap. Regions only overlap if they overlap in all dimensions of the namespace. The RTI will then deliver the publication to all subscribers with overlapping region.

- If a federate subscribes/publishes without region, the RTI will assume that the federate wishes to subscribe/publish in the entire namespace.

### 2.2.7 Support services

The MOM part of the FED as introduced during the discussion of the OMT, provides useful support services for federates. Two samples are given below:

- Handle request services. Each instance in the HLA, whether it’s an object, attribute, interaction, federate or namespace, is characterized by a numeric handle. The handle request services provide these handles based on names of the instance.

- Report services. The RTI can be instructed to provide detailed reports on participating federates.

### 2.3 HLA Rules

The final part of the HLA specification is formed by the HLA rules document ([DOD98_R]). The summary of the document is given below, without comment:

**Rules for federations:**

1. Federations shall have an HLA federation object model (FOM), documented in accordance with the HLA Object Model Template OMT.

2. In a federation, all simulation-associated object instance representation shall be in the federates, not in the runtime infrastructure (RTI).

3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.

4. During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification.

5. During a federation execution, an instance attribute shall be owned by at most one federate at any given time.

**Rules for federates are:**

1. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the
HLA OMT.

2. Federates shall be able to update and/or reflect any attributes and send and/or receive interactions, as specified in their SOMs.

3. Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOMs.

4. Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes, as specified in their SOMs.

5. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

3

HLA made practical

In this chapter, a brief survey of RTI-tools available will be discussed. In the first paragraph, a typical implementation, based on Pitch AB’s portable RTI (http://www.pitch.se), will be discussed.

3.1 A typical implementation

Pitch AB’s portable RTI (pRTI) is commercial, HLA 1.3 compliant, RTI software. The Exploration version is based on the Java API and can only be used in conjunction with federates that implement the Java HLA API. However, the enterprise edition of pRTI supports a C++ API as well. A generalized architecture of the pRTI implementation is shown in Illustration 6.1.12 above.

The two objects that actually implement the HLA interfaces (RTI ambassador & federate ambassador) are:
• A federate ambassador object (local on the federate’s machine)
• An RTI ambassador object (also local on the federate’s machine, with a remote connection to the RTI).

These objects are created the moment the federate registers with the RTI. The illustration also shows an a-symmetry between requests from federates to RTI versus requests from RTI to federate:
• Requests from federate to RTI take place via the local RTI ambassador, that implements the RTI-ambassador interface. The RTI ambassador then relays the request to the remote RTI implementation.
• Requests from RTI to federate take place via a remote invocation of the RTI ambassador. The RTI ambassador then locally relays the request to the federate ambassador that finally relays the request to the federate.

However, this is only one of many implementations that are possible, since the main restriction posed by the HLA specification is that the two HLA interface are implemented, but is doesn’t specify how this should done.

3.2 Support tools available
Since the start of the HLA specification process, awareness has grown that the HLA should be well supported by software tools. The Defense Modeling and Simulation Office has therefore set up the Software Distribution Center (http://www.sdc.dmoso.mil) where the following support tools can be downloaded:
• Run-Time Infrastructure Next Generation (RTI-NG)
• Run-Time Infrastructure Performance Benchmarks
• Federation Execution Planners Workbook Editor (FEPW)
• Object Model Development Tools (OMDT)
• Federation Verification Tool (FVT)
• Federation Management Tool (FMT)
• Data Collection Tool (DCT)

4 Conclusion
Eventtypes & interfaces between simulation and Java

1 Sequence diagrams

Illustration 6.1.16: Sequence diagram of event broadcast
2 IDL interface specification

/**
 * Interface: NARAD_Passport
 *
 * Description: Interface for an object that can identify a
 * NARAD-device within the current domain.
 */

interface NARAD_Passport
{
    long getId();
    void setId(in long inID);
    string getType();
    void setType(in string inSource);
};
interface NARAD_Event
{
    NARAD_Passport getSource();
    void setSource(in NARAD_Passport inSource);
    NARAD_Passport getDestination();
    void setDestination(in NARAD_Passport inSource);
};

interface NARAD_PropertyChangeEvent:NARAD_Event
{
    double getValue();
    void setValue(in double value);
    string getUnit();
    void setUnit(in string unit);
    string getProperty();
    void setProperty(in string property);
};

interface NARAD_StateChangeEvent:NARAD_Event
{
    string getState();
    void setState(in string state);
};
interface NARAD_CORBA_Server
{
    string hello(in string visitor);
    void notifyEvent(in NARAD_Event evt);
    boolean hasEvent();
    NARAD_Event getNextEvent();
};
3 Event model & translations

Illustration 6.1.18: Event types used in the architecture

Events of type B are delivered as eventtype A to process A, translation takes place in process B.

Events of type A are delivered as eventtype A to process B, translation to eventtype B takes place in process B.

Conclusion: all translation actions take place in the lower process (here process B).
The following requirements have been imposed on simulating in the Arena environment, in order to be able to use the developed CorbaDeviceWrapper to communicate via CORBA and Java with the architecture:

1. Each object class, for which property change are broadcast to the environment, is specified by a specific variable, having the class's name as name, and the number of instances as value. An example is the ‘Carwash’ variable in the illustration below, actually having the value 2.

2. Each property has its own variable, named as <class> + <property>. The soapLevel property of the carwash therefore is therefore named in Arena as 'carwashSoapLevel'. Each property variable keeps the property values for all the class's instances. The 'carwashSoapLevel' property thus keeps the soapLevel property values for the two defined carwashes.

3. Event properties of type string (for instance the 'unit' property of a propertyChangeEvent), cannot be directly set to a string value in Arena. Instead, a variable having the name of the string value needs to be used, which value can be referred to by using Arena’s NSYM command.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Basic Process</th>
<th>Rows</th>
<th>Columns</th>
<th>Clear Option</th>
<th>Initial Values</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gallon</td>
<td></td>
<td></td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>carwash</td>
<td></td>
<td></td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>carwashPowerUsage</td>
<td>2</td>
<td>1</td>
<td>System</td>
<td>2 rows</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>carwashSoapLevel</td>
<td>2</td>
<td>1</td>
<td>System</td>
<td>2 rows</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>carwashMaxLevel</td>
<td>2</td>
<td>1</td>
<td>System</td>
<td>2 rows</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>carwashState</td>
<td>2</td>
<td>1</td>
<td>System</td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>maxCarwashStates</td>
<td>2</td>
<td>1</td>
<td>System</td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>GasStation</td>
<td>5</td>
<td></td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>maxGasStationStates</td>
<td>5</td>
<td>1</td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MultiSet</td>
<td>0</td>
<td></td>
<td>System</td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>iWatt</td>
<td>0</td>
<td></td>
<td>System</td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ToExternals</td>
<td>0</td>
<td></td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>GasStationSoapStock</td>
<td>4</td>
<td>1</td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>GasStationWaxStock</td>
<td>1</td>
<td>1</td>
<td>System</td>
<td>1 rows</td>
<td></td>
</tr>
</tbody>
</table>

Illustration 6.1.19: Screenshot of variables in a wrapped Arena simulation