Towards the agile enterprise

A method to come from a DEMO model to a Normalized System,
applied to Government Subsidy Schemes

MARIEN KROUWEL, B.SC.

December 15, 2010

Towards the agile enterprise

A method to come from a DEMO model to a Normalized System,
applied to Government Subsidy Schemes

THESIS

submitted in partial fulfilment of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Marien Krouwel, B.Sc.
born in Utrecht, The Netherlands

TU Delft
Delft University of Technology

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Universiteit
Antwerpen

Abstract

Because of the changing needs of enterprises, raised by the changing market among other reasons, the needs of its supporting information system(s) will also change. Design and Engineering Methodology for Organizations (DEMO), from Delft University of Technology, has already proven to be an effecitve tool in designing and realizing agile organizations. The next step is to have these organizations supported by agile Information and Communication Technology (ICT) systems. The Normalized System (NS) approach, from the University of Antwerp, seems to be key for developing agile ICT systems.

In this masters thesis, it is investigated whether and how DEMO and NS could be combined optimally in order to cover the whole development process of agile enterprises, starting from a DEMO model and arriving at an implemented organization, with its supporting ICT systems. To make the project as concrete as possible, Government subsidy schemes from Capgemini is used as a practical case for clarification and demonstration.

Contact Information

Author

Name	Marien Krouwel
Student number	1552880
E-mail	marienkrouwel@gmail.com

Research group

Department of Software Techonology
Faculty of EEMCS
Delft University of Technology
Delft, The Netherlands
http://www.ewi.tudelft.nl/

Company

Capgemini Netherlands, Utrecht, Papendorp
Division Technology, Sector Public
Papendorpseweg 100, 3528 BJ
Utrecht, The Netherlands
http://www.nl.capgemini.com/

In collaboration with research group

Department of Management Information Systems
Faculty of Applied Economics
University of Antwerp
Antwerp, Belgium
http://www.ua.ac.be/main.aspx?c=.ZOEKEN&n=37738&afdCode=UA042

Thesis committee

Prof. Dr. Ir. Jan L.G. Dietz (chair), j.l.g.dietz@ewi.tudelft.nl, Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Department of Software Techonology

Ir. Hans J. Geers (member), h.j.a.m.geers@tudelft.nl, Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Department of Software Techonology

Dr. Joseph Barjis (member), j.barjis@tudelft.nl, Delft University of Technology, Faculty of Technology, Policy and Management, Department of Multi Actor Systems

Prof. dr. Jan Verelst (member), jan.verelst@ua.ac.be, University of Antwerp, Faculty of Applied Economics, Department of Management Information Systems

Dr. Martin J. Op 't Land (external), martin.optland@capgemini.com, Capgemini Netherlands, Utrecht, Papendorp

PREFACE

This thesis has been submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, track Information Architecture. The Information Architecture master track is a collaboration between the faculty Electrical Engineering, Mathematics and Computer Science (EEMCS) and the faculty of Technology, Policy and Management (TPM), both of the Delft University of Technology. In this master track, I have gained a lot of knowledge by learning from two different faculties with different expertises.

This research is executed at Capgemini Netherlands N.V. from January until September 2010. I am very thankful to Capgemini, in particular to Martin Op 't Land, who gave me the opportunity to gain practical experience at a company with much expert knowledge in a lot of areas that are also thought within the master track Information Architecture.

Normally one starts a an internship at a company by meeting the people and reading into the subject matter. I, however, started my internship with being ill after which I had to spent a week in an hospital because of an appendicitis. During the one month rehabilitation that followed, I was supported by both my family and Capgemini, who all made it so easy for me. However, I was glad I could re-start my internship half of March.

There are many people who I would like to thank for educating and supporting me throughout this thesis research. First, I would like to thank Jan Dietz for providing me better insights in DEMO, for the feedback he gave me on my literature research and thesis research, and for the guidance on scientific grounds during my thesis. Also thanks go to Jan Verelst and Herwig Mannaert from the University of Antwerp for the discussions we had, which gave me better insight in the Normalized System theory, and for inspecting the final outcome of this research. And thanks go to Hans Geers and Joseph Barjis for being part of my thesis committee and reviewing my research.

In particular I am thankful to Martin Op 't Land who not only introduced me to Capgemini but also introduced me to the topic of Normalized System. We also had many discussions which provided me better insight in any of the topics covered in this research. But next to that he also acted as a friend who supported me when I was struggling, not only in performing my research or writing my thesis, but also in other aspects of life. Also thanks to other employees of Capgemini, Kees Nootenboom, Jan van Santbrink, Cor Nagtegaal, Johan van der Graaf, among others, with whom I have had nice discussions and who showed me what kinds of business are performed at Capgemini.

Last, but definitely not least, I would like to thank my family for their support in writing this thesis, my friends for joining me for a cup of coffee when not writing my thesis, and Eveline, friend for life, for her support in everything, even in spending less time together.

Marien Krouwel
December 15, 2010

CONTENTS

List of Figures

List of Tables

LISTINGS

1. INTRODUCTION

Enterprises are always in the search of new opportunities, arising from changing market needs, to stay ahead of competition. However, accommodating such a business change is often not straightforward. A large time-to-market could even mean a competitor takes the opportunity first. And even then, often the change should still be made in order to remain competitive.

Most enterprise are supported in their operations by information systems. Already, information systems often do not meet the business requirements and thus do not support the enterprise fully [Cap07,Bar08]. This is partly caused but also deteriorated by the fact that enterprises change. In this research it is tried to find a new way of creating an information system that is able to support a changing enterprise.

1.1 Problem statement

Agility is defined as being flexible and adaptable. An agile enterprise is an enterprise that is able to adapt rapidly and cost efficiently in response to changes in the environment [TV02]. Examples of changes an enterprise has to deal with are [AEOtL00]:

- change of business (adding a service) to meet changing market needs or to remain competitive;
- change of rules, e.g. a law change;
- change of processes to lower costs or improve quality;
- change of technology, e.g. adding a new way for clients to communicate with the enterprise, using machines instead of humans for some task, and even outsourcing.

As stated before, most enterprises are supported by information systems. Enterprises use information systems since they speed up processes by taking over tasks from human beings - imagine the time it would cost for an accounting office to perform the bookkeeping for a large enterprise without the use of a computer or calculator. Speeding up processes is necessary for enterprises to work effeciently and to be more cost effective. If an enterprise wants to change but also relies on an information system, that system has to change as well. An information system that can be adapted to such changes, is called an agile information system. It can be concluded that for an enterprise to be truly agile, it needs to be supported by an agile information system. This research aims at modeling an agile information system that supports the agile enterprise.

1.2 Research goal and approach

With DEMO, from Delft University of Technology, the essential business processes of an enterprise can be described in a coherent, consistent, comprehensive and concise way. DEMO has

already proven to be an effective tool in realizing agile enterprises[1]. The NS approach, from the University of Antwerp, seems to be key for developing agile information systems. Therefore, it is considered interesting to explore how DEMO and NS can be combined to create a information system that supports an enterprise and is able to adapt to the changing needs of an enterprise. The research goal can now be formulated:

> *Construct a method to create a Normalized System that supports an enterprise modelled with DEMO.*

In order to find such a method, the theories underlying the methods must first be matched with each other; if DEMO and NS theories don't match, it might be difficult to realize an agile information system in this way. Then it must be investigated what specifications are needed to create a NS. These specifications then must be found in a DEMO model and it must be investigated whether information is missing in a DEMO model to create a NS. These steps are summarized in Table 1.1. Only if these four steps are completed, the desired method can be composed. To find whether the designed method really works, and to find any problems that could arise when applied to a practical case, the case Government Subsidy Schemes (in Dutch: Rijkssubsidieregelingen) from Capgemini is used for clarification and demonstration.

Research question	**Answering method**
1. Do the DEMO and NS theories match?	Analysis and comparison of both theories
2. How to model a Normalized System?	Analysis of the NS approach
3. How to extract specifications for a NS from a DEMO model?	Analysis of DEMO modeling technique
4. What information is missing in a DEMO model for a complete set of specifications for a NS?	NS model completeness check

Table 1.1: Research questions and answering methods

In practice, the different steps were performed iteratively; by applying the found method to the Subsidy Scheme case, design flaws in the method and even theoretical differences were found as well as ways to improve the method. In Section 2.3, the research approach is placed in theGeneric System Development Process (GSDP).

1.3 Structure

In Chapter 2, DEMO and its underlying ψ-theoryare described. Also, a brief elaboration of the GSDP is provided and the relation between this research and the GSDP is shortly outlined. In Chapter 3, the Normalized Systems approach is described. Since the NS approach does not provide a modeling technique, one is proposed, already answering question 2. In Chapter 4, it is investigated if the two theories can be combined and what conflicts could arise, answering question 1. In Chapter 5, it is tried to model a NS from a DEMO model, answering question 3. In that chapter is also investigated whether any specifications are missing in a DEMO model

[1]For examples of case studies, the reader is referred to http://www.demo.nl/blogcategory/practical-case-studies/

for a complete set of NS specifications, answering question 4. In Chapter 6, finally, the method, composed from the findings from the earlier chapters, is presented. In Chapter 7, the gained method is applied to the case Government Subsidy Schemes from Capgemini for demonstration. This research concludes with a summary of the results as well as their implications and limitations and a set of of ideas for future work (Chapter 8).

1.4 Reading guide

If you are interested in...	**...you should read...**
DEMO	Chapter 2
Normalized System	Chapter 3
How to model a NS	Section 3.2
Theoretical comparison	Chapter 4
The method for extracting NS elements from a DEMO model	Chapter 6
Example of applying the method	Chapter 7
Some important conclusions	Chapter 8

Part I

Theoretical Foundation

2. DEMO

DEMO is a methodology for the design, engineering, and implementation of organizations and networks of organizations [Sti]. DEMO was developed by Prof. Jan Dietz at Delft University of Technology. One of the key elements of DEMO is that it brings forth the essence of an organization fully independent from the way in which it is realized and implemented. This is called an enterprise ontology. Dominant in the DEMO methodology is the ψ-theorythat underlies this notion of enterprise ontology. The ψ-theory(way of thinking) is elaborated on in Section 2.1 and DEMO (way of modeling) is shown in Section 2.2. In Section 2.3, the Generic System Development Process for is presented and this research is placed in the GSDP. Only the parts relevant for this research are discussed, for further reading the reader is referred to Dietz [Die06, Die08].

2.1 Ψ-theory

The ψ-theory- $\psi$is pronounced as PSI and stands for Performance in Social Interaction - states that an organization consists of people who, while communicating, enter into and comply with commitments (social interaction) about the things they bring about in reality (performance) [DH04]. The ψ-theoryconsists of four axioms and one theory, which will be explained in the next sections.

2.1.1 Operation axiom

The operation axiom states that the operation of an organization is constituted by the activities of actor roles (elementary chunks of authority and responsibility) fulfilled by subjects. An actor (a subject fulfilling an actor roles) can perform two kinds of acts: production acts (P-acts), and coordination acts (C-acts). Both production and coordination acts have definite results, namely production facts (P-facts) and coordination facts (C-facts), respectively. By performing P-acts, actors contribute to bringing about goods or services or information or data that are delivered to other actors, inside or outside the organization. By performing C-acts, actors enter into and comply with commitments towards each other regarding the performance of P-acts. A C-act is always performed by one actor role, called the performer (P), and directed to another actor role, called the addressee (A). Both actor roles may be fulfilled by the same actor, or subject.

2.1.2 Transaction axiom

The transaction axiom states that coordination acts are performed as steps in universal patterns. These patterns, also called transactions, always involve two actor roles: an initiator and an executor. They are aimed at achieving a particular result, the P-fact. A transaction evolves in three phases: an execution phase, enclosed by two conversation phases. The first phase is the order phase in which the initiator and the executor work to reach an agreement about the intended result of the transaction, i.e., the production fact that the executor is going to create

as well as the intended time of creation. The second phase is the execution phase in which this production fact is actually brought about by the executor. In the third phase, the result phase, the initiator and the executor work to reach an agreement about the production fact that is actually produced, as well as the actual time of creation. Only if this agreement is reached will the production fact come into existence.

Although Dietz states there are many different intentions for C-acts [Die06, p.83], for the basic transaction pattern, only request, promise, state, and accept are used. This is the course that is taken when the initiator and the executor keep consenting, see Fig. 2.1. However, they may also dissent. There are two states where this may happen, namely the states "requested" and "stated". Instead of promising one may respond to a request by declining it, and instead of accepting one may respond to a statement by rejecting it. It brings the process in the state "declined" or "rejected" respectively. These states are indicated by a double disk, meaning that they are discussion states. If a transaction ends up in a discussion state, the two actors must 'sit together', discuss the situation at hand, and negotiate about how to get out of it. The possible outcomes are a renewed request or statement (probably with a modified proposition) or a failure (quit or stop). The basic pattern extended with these negotiation and failure states is called the standard transaction pattern, see Fig. 2.2.

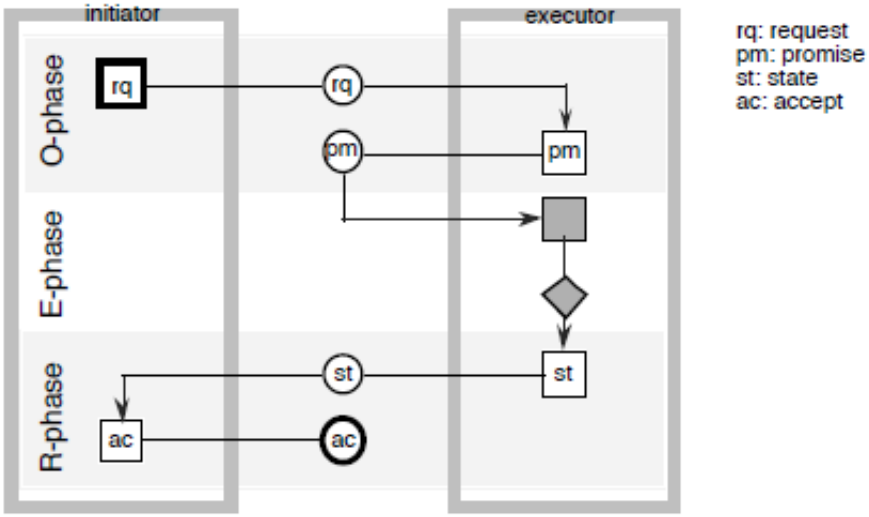


Figure 2.1: Basic transaction pattern [Die06]

In practice, it is quite common that either the executor or the initiator wants to revoke an act. This may result in a partial or complete rollback of the transaction [Die06]. This is accomodated by the option to cancel any C-act in the basic pattern at any time. This gives rise to four cancellation patterns which are all shown in Fig. 2.3. Every cancellation starts with a cancel act on which a conditional C-fact is put, meaning the C-fact should come into existence before it can be cancelled. After having received the request for cancellation, the other party has to decide whether he refuses or allows the cancellation. If he refuses, the standard pattern is continued (if it wasn't finished yet). If allowed, another act of the standard pattern will follow, constituting a new fact. Moreover, a rollback action could be started but that is not part of the transaction pattern.

A C-act can be performed in three different ways: explicitly (by some observable act), implicitly (performing one C-act also counts for performing another one), or tacitly (there is no

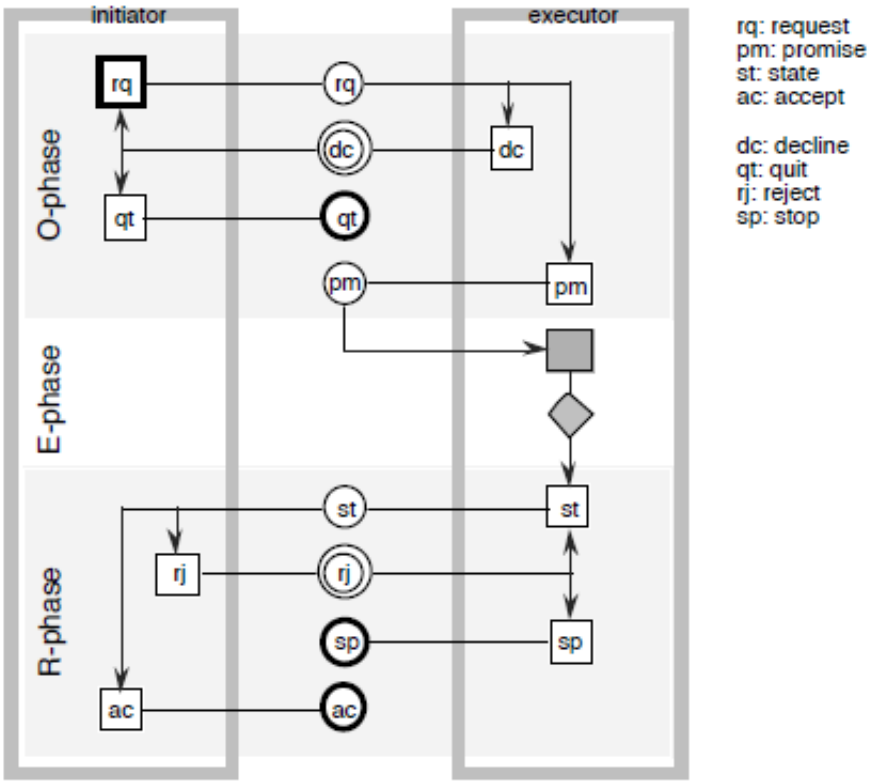


Figure 2.2: Standard transaction pattern [Die06]

observable act that could count as performing the C-act). Moreover, a C-act can be delegated to some other actor (role).

2.1.3 Composition axiom

The composition axiom states that every transaction is enclosed in some other transaction, or is a customer transaction of the organization under consideration, or is a self-activation transaction. By enclosing transactions in another transaction, P-facts are interrelated. Most business processes don't consist of one (atomic) act, they are composed of different acts, like an assembly line. For a government to provide a subsidy, first the subsidy application has to be checked both formally and substantively (resulting in a grant), then the amount of subsidy has to be calculated, and then the money has to be payed out. In terms of transactions, this would mean the check transactions and pay transaction are enclosed in the customer-initiated transaction for giving subsidies (Fig. 2.4).

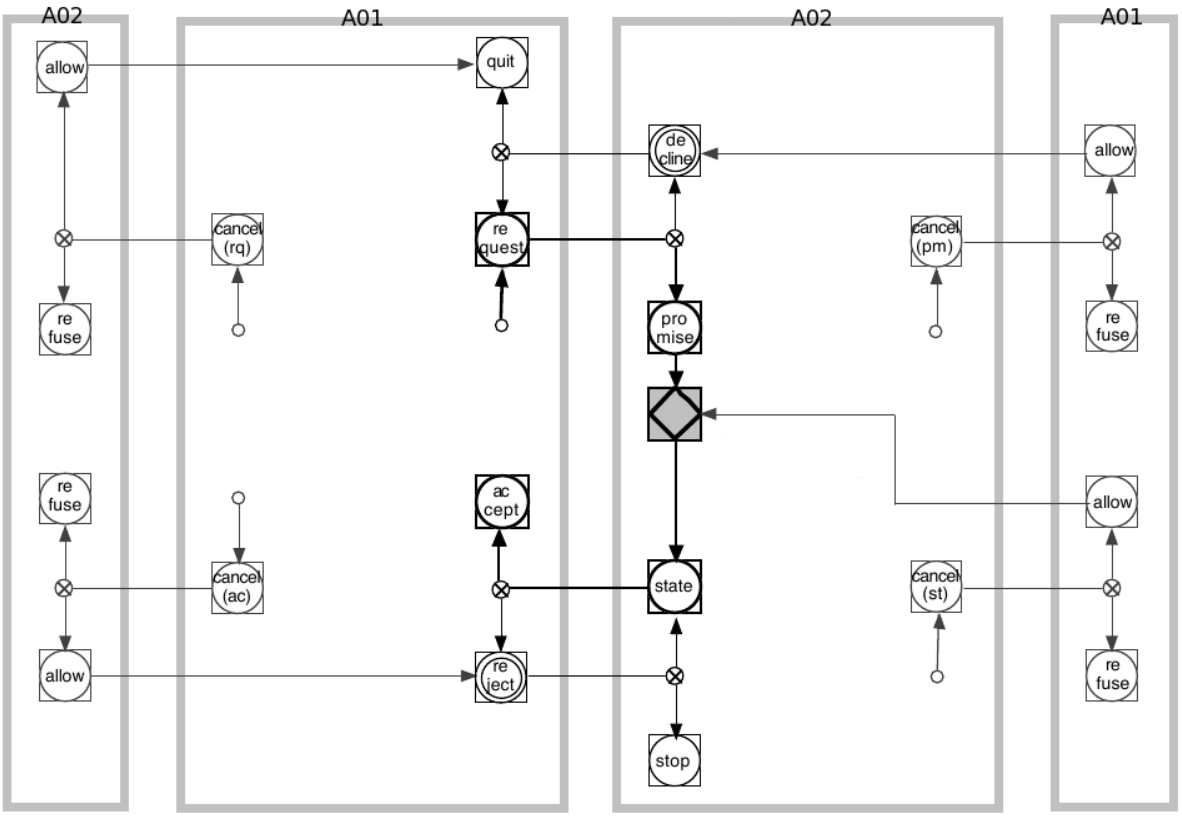


Figure 2.3: Complete transaction pattern [Die08]

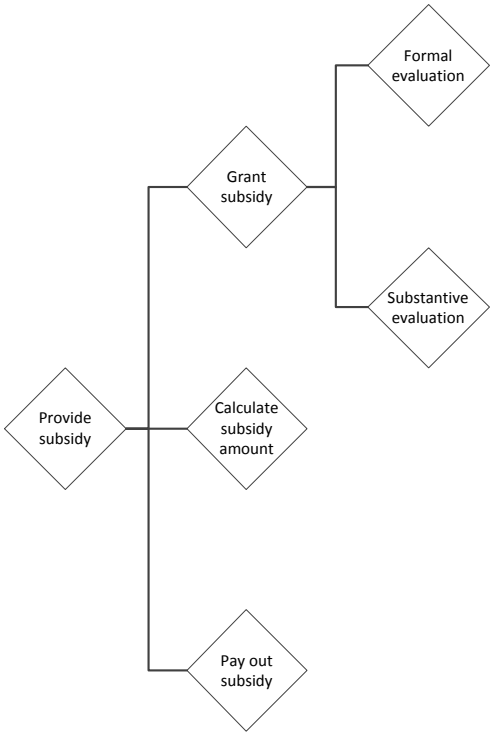


Figure 2.4: Component structure of a (governmental) subsidy agency

2.1.4 Distinction axiom

The distinction axiom states that there are three distinct human abilities playing a role in the operation of actors, called performa, informa and forma. Those abilities are recognized in both coordination and production acts (see Fig. 2.5).

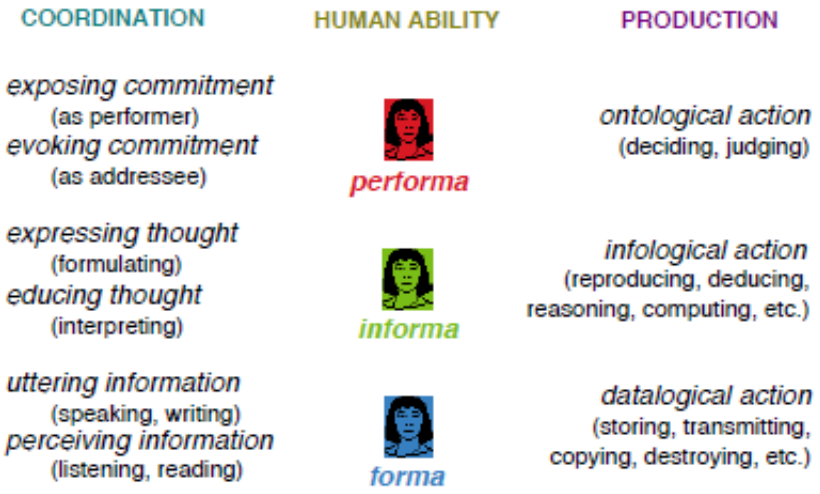


Figure 2.5: Summary of the distinction axiom [Die06]

Regarding P-acts, the forma ability is the ability to store, retrieve, and transmit data or documents. These acts are all called datalogical acts and are all actions by which the content of the data or documents is of no importance. actors which use the forma ability to perform P-acts are called documental actors (D-actors). Similarly, a transaction in which a datalogical act occurs, is called a D-transaction.

The informa ability is the ability to reason, compute, and to remember and recall knowledge. These acts are called infological acts and are all actions by which the content data or documents, abstracted from the form aspects, is of importance. actors which use the informa ability to perform acts are called intellectual actors (I-actors). A transaction in which an infological act occurs, is called an I-transaction.

The performa ability is the ability to establish new, original things. Examples of performa ability are engaging into commitments, deciding, and judging, together called ontological acts. The performa ability is considered the essential human ability for doing business. actors which use the performa ability to perform P-acts are called business actors (B-actors). A transaction in which an ontological act occurs, is called a B-transaction.

As stated by the operation axiom (Section 2.1.1), by performing C-acts, actors enter into and comply with commitments towards each other with respect to the performance of P-acts. At this point, the P-act can be of any level, i.e., datalogical, infological, or ontological. In order to successfully perform a C-act, different kinds of communicative acts have to be performed, corresponding with the three human abilities as shown in Fig. 2.5. In order to have a C-act from P to A performed successfully, a performative exchange between P and A must take place in which P exposes his commitment in a performative act, addressed to A, and in which A has to evoke in her the corresponding commitment. However, the only way of P to expose its commitment and to make it knowable to A, is to express it through its informa ability, followed by the inducement in the mind of A of an equivalent thought, by means of its informa ability. The

intellectual understanding between P and A comes into existence by an informative exchange. Expressing a thought can only be done by formulating it in a sentence in some language, and at the same time uttering it in some form, such as speaking or writing. Significational understanding between P and A only comes into existence by a formative exchange by P and A in their forma ability. The process of performing a C-act is exhibited in Fig. 2.6. So, during aC-act both the performer and the addressee shape between abilities several times. B-actors, I-actors and D-actors only distinguish themselves by the kind of production act [dJ09].

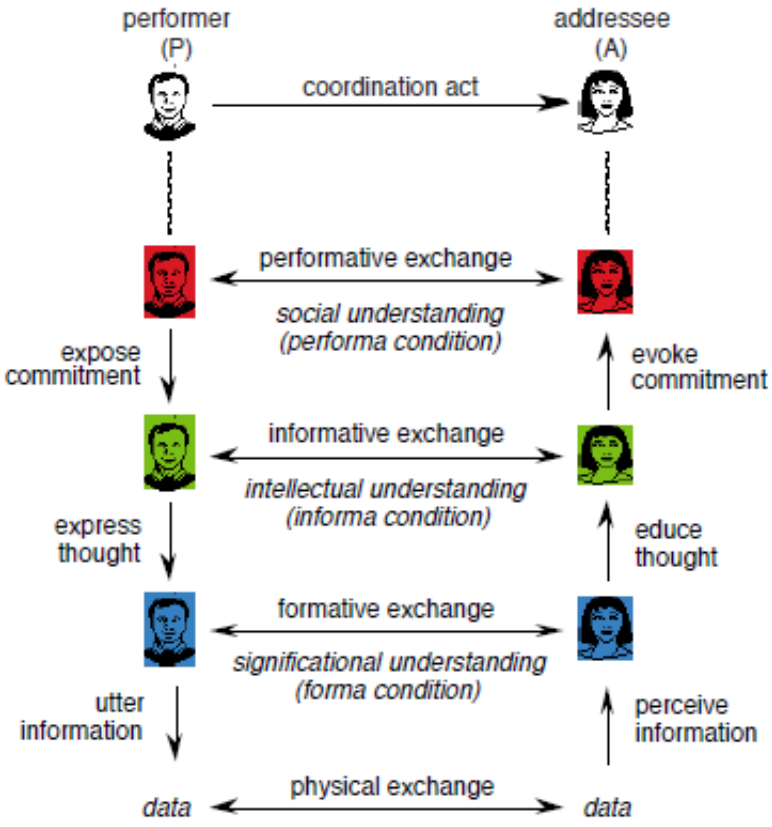


Figure 2.6: The process of performing a coordination act [Die06]

2.1.5 Organization theorem

The organization theorem states that the organization of an enterprise is a heterogeneous system that is constituted as the layered integration of three homogeneous systems: the B-organization (B from Business), the I-organization (I from Intellect), and the D-organization (D from Document).The relationships among them are that the D-organization supports the I-organization, and the I-organization supports the B-organization. These relationships are established through the human being possesing the three distinct abilities (Section 2.1.4).

The integration of the three aspect organizations is called the realization of an enterprise. Puspa Sandhyaduhita [San09], and Joop de Jong and Jan Dietz [dJD10], have ellaborated on how to define the I- and D-organization from the B-organization, based on the GSDP (Section 2.3).

2.2 Aspect models

The complete model of the B-organization of an enterprise in DEMO provides a complete and essential enterprise ontology consisting of four related aspect models as exhibited in Fig. 2.7:

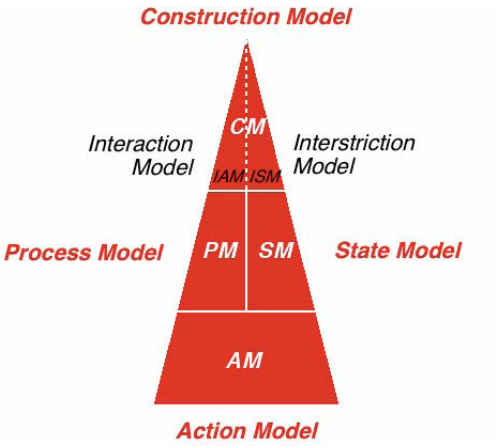


Figure 2.7: The ontological aspect models [Die06]

2.2.1 Construction Model

The Construction Model (CM) specifies the composition, environment, and construction of an organization; it shows the identified transaction kinds and associated actor roles as well as the information banks and connections between these banks and actor roles. The CM is the most concise model of an organization. This is shown in the figure as it occupies the top of the triangle; there is nothing above it.

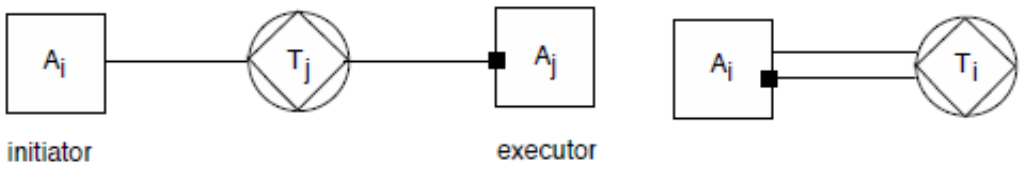
The CM is split up into two models again: the Interaction Model (IAM) and the Interstriction Model (ISM). The IAM of an organization consists of the transaction kinds in which the identified actor roles participate as the initiator or executor. It shows the active influences between actor roles: the execution of transactions. The IAM is expressed in an Actor Transaction Diagram (ATD) and a Transaction Result Table (TRT). The ISM shows the passive influences between actor roles, i.e., the taking into account by an actor role of existing facts when being active. Facts are either of the type production or coordination, both can be found in their own designated production and coordination bank respectively. The ISM is expressed in an Actor Bank Diagram (ABD) and a Bank Contents Table (BCT). Together the ATD and ABD form the Organization Construction Diagram (OCD), so the CM can also be expressed in only an OCD, TRT and BCT. Therefore, the ATD and ABD are not elaborated on.

In the OCD, a square denotes an actor role. Two kinds of actor roles are distinuished: elementary (white) and composite (gray). Moreover, there is a boundary dividing the set of actor roles within the kernel of an organization (composition), and the set of actor roles outside the organization (environment). By convention, all actor roles in the environment are composite actor roles. The reason for this is that it doesn't matter whether an external actor role is elementary or composite, the interest is in the kernel of the organization [Die06, p.159].

A diamond represents a production bank (storing production facts), and a circle denotes a coordination bank (storing coordination facts). The combination of a production and coor-

dination bank is called an information bank or transaction and is represented by a diamond within a circle. As with actor roles, there are elementary and composite banks; a bank is composite if it is unknown whether it is an elementary bank or if it is the union of two or more elementary banks. As was the case for composite actor roles, external banks are considered to be composite, and composite banks are shaded gray.

A dotted line between an actor role and an information bank denotes an information link meaning the actor role needs to inspect that bank in order to follow up his agenda. Every transaction has two outgoing lines: one for the initiator and one for the executor; the executor link has an extra black square. Either, the initiator and executor of a transaction are two different actor roles (Fig. 2.8a) or they are the same (Fig. 2.8b). This last construct is known as self-activation (see Section 2.1.3). The complete legend of the OCD is shown in Fig. A.1.



(a) Transaction with initiator and executor (b) Self-activation

Figure 2.8: Basic constructs of the ATD

Often, first a global ATD is made, representing the kernel of an organization by a single composite actor role. This is convenient as it clearly shows to the outside world what the business is about; internal details are hidden within the kernel. For the subsidy agency this is shown in Fig. D.1. When the kernel is opened and split into elementary actor roles, often new (internal) transactions are revealed.

The TRT is simply a table demonstrating the identified transaction kinds and result kinds. For the subsidy agency, the TRT is shown in Table D.1.

A BCT specifies the fact banks in which the elements of object classes, and the instances of fact kinds and result kinds from the Fact Model (FM) (Section 2.2.4) are contained. Since result kinds and fact kinds are only the result of production and not of coordination acts, the BCT only lists production banks. Some of those production banks come with a coordination bank (transactions), others don't (external fact banks). With this table, it can easily be checked whether an actor role has access to the information an actor role needs.

2.2.2 Process Model

The Process Model (PM) contains all identified process steps in all identified transaction kinds and the existence laws that apply. The PM also contains the occurence laws that hold for the transaction steps, including the cardinalities of the occurrences. There are two kinds of occurrence laws: precedence laws (causal conditions) and prerequisite laws (wait conditions). The PM is put just below the CM in the triangle because it is the first level of detailing of the CM, namely, the detailing of the identified transaction kinds.

The PM is represented by a set of Process Structure Diagrams (PSDs) and Transaction Pattern Diagrams (TPDs). In these diagrams, precedence laws are represented by solid arrows,

whereas prerequisite laws are represented by dashed arrows. The legend of the PSD is shown in Fig. A.2. The rounded rectangle is an transaction symbol in which the diamond remains unchanged. There is an invisible, non-proportional, time line from left to right. Every transaction status may, except the being executed, serve as a point from which another transaction is initiated. Likewise, every transaction status, except the being executed, may serve as a point from which a wait condition holds. Moreover, there may be a wait condition for every transaction step [Die09a]. The PSDs for the subsidy agency are shown in Fig. D.3.

The TPDs in fact constitute for each transaction the complete transaction pattern (see Section 2.1.2). However, for clarification, inter transaction links may be added. Since every transactions knows the complete transaction pattern, and the inter transaction links can also be seen in the PSDs, in this research it is considered the TPDs do not provide any new information and therefore can be left out.

2.2.3 Action Model

The Action Model (AM) specifies the action rules that serve as guidelines for the actors in carrying out their transactions. The AM is put just below the PM because it is the second level of detailing of the CM, namely, the detailing of the identified steps in the PM of the transaction kinds in the CM; the AM is in a very literal sense the basis of the other aspect models since it contains all information that is (also) contained in the CM, PM, and FM. At the ontological level of abstraction there is nothing below the AM.

The AM is represented by means of Action Rule Specifications (ARS). Basically, it contains an action rule for every agendum kind for every identified actor role. The basic syntax can be found in Listing A.1. The ARS and thus the complete AM for the subsidy agency can be found in Listings D.1-D.7.

2.2.4 Fact Model

The FM of an organization is the specification of the state space and the transition space of the production world of the organization. The FM contains the identified transaction result kinds, fact kinds, and object classes (extensions of unary fact kinds; also called entity types) as well as the existential laws that apply. Four kinds of laws are distinguished: reference laws, unicity laws, dependency laws, and exclusion laws. Occurrence laws between transaction result kinds can be shown by declaring the one as a specialization of the other [Die06, Die09a]. The FM is put on top of the AM because it is directly based on the AM; it specifies all object classes, fact kinds, and ontological coexistence rules that are contained in the AM. On the other side, the FM is put just below the CM, because it may also be viewed as another detailing of a part of the CM, namely, the contents of the information banks.

An FM is expressed in a State Space Diagram (SSD) and a Fact Definition List (FDL). The SSD is fully based on the language World Ontology Specification Language (WOSL), a language for the specification of world ontologies. Because of the similarity between the ontology of a world and the conceptual schema of a database, Object Role Modeling (ORM) is used as basis for the graphical notation. WOSL is described in e.g. [Die06] and ORM in e.g. [Hal09]. The complete legend of the SSD is shown in Fig. A.3. The SSD of the subsidy agency is shown in Fig. D.4.

The FDL is simply a list of derived facts and their derivation rule, a convenient way of specifying fact kinds that are proper (mathematical) functions, and of which the range is a set of values. One may as well specify them in the SSD, but that would make the SSD unnecessarily voluminous. The fact kinds in an FDL are also called properties (of object classes) [Die06,Die09b]. The FDL for the subsidy agency is shown in Listing D.9.

2.3 Generic System Development Process (GSDP)

By system development is meant the bringing about of a new system or of changes to an existing system. The system development process comprises all activities that have to be performed to arrive at an implemented system. Any system development process concerns two systems: the Object System (OS), the system which is going to be developed, and the Using System (US), the system that is going to use the functionalities, or services, of the OS. The complete process, as shown in Fig. 2.9, consists of three phases: design, engineering, and implementation.

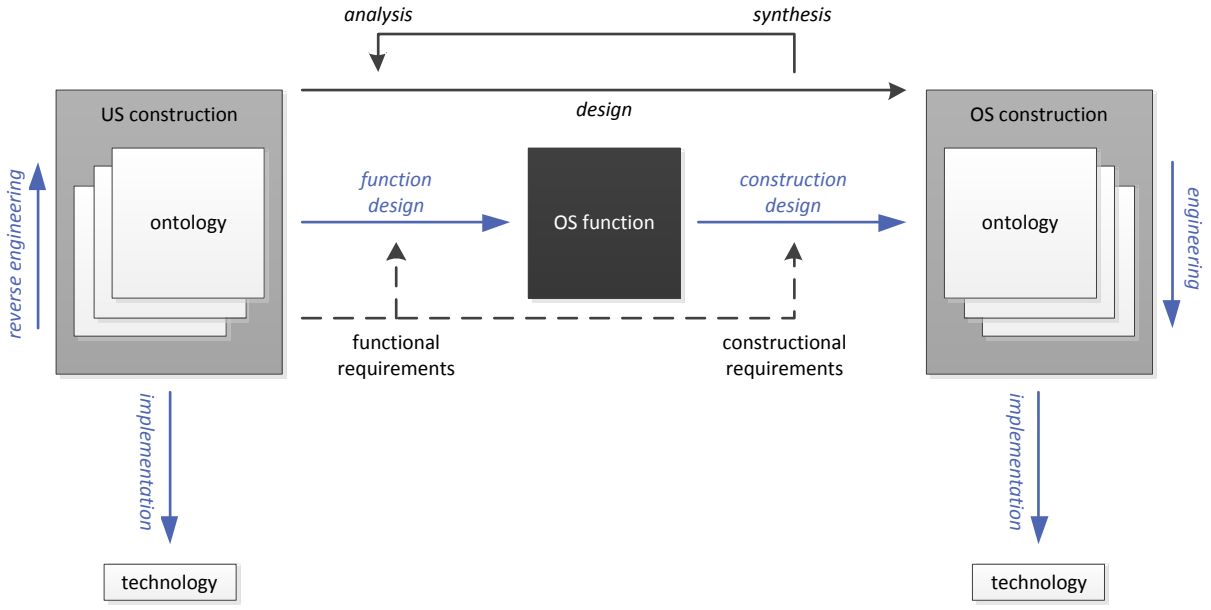


Figure 2.9: The Generic System Development Process (GSDP)

The design phase is further split up into function design and construction design. Function design starts from the construction of the US and ends with the function of the OS. The result is a functional (black box, see Dietz [Die06, Die08]) model containing all functional specifications. In construction design, the specifications are devised. Construction design starts with the specified function of the OS and ends with the construction of the OS. The total design of a system is understood as a process of alternate analysis and synthesis steps. An analysis step is one in which the problem is better understood; a synthesis step is one in which the solution becomes more clear.

The engineering of a system is the process in which a number of subsequently more detailed constructional (white box, see Dietz [Die06,Die08]) models are produced such that every model is fully derivable from the previous one and the available specifications. Engineering starts from the ontological model and ends with the implementation model. In practice, it is often the

case that one or more constructional models are missing and only the implementation model is present. The process of constructing the ontological model is known as reverse engineering.

By implementing a system is understood the making operational of the organizations' realization by the assignment of technological means to the constructional elements in the implementation model. Examples of technologies are the utilization of human beings, and of ICT. Different technologies can be used together. The utilization of ICT is elaborated on later. Following the operation axiom (Section 2.1.1), three kinds of technologies are distinguished: actor technology, coordination (communication) technology, and production technology.

The development process of the I-organization can be considered as an instance of the GSDP [dJD10, Die08]. The B-organization is now the US and the I-organization is now the OS. It is concluded that it is the the *function* of the I-organization that supports the *construction* of the B-organization. Similarly, the function of the D-organization supports the construction of the I-organization. This is summarized in Fig. 2.10.

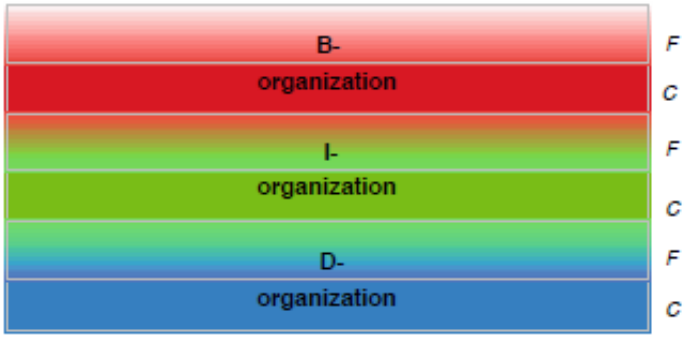


Figure 2.10: The layered integration of an organization [Die06]

2.3.1 The position of ICT in an organization

The relation between organization and ICT can now be given by combining the GSDP and the layered integration of the three organizations (Section 2.1.5 and Fig. 2.10), see Fig. 2.11. For each of the aspect organizations, applications can be engineered. D-applications, such as text processors, operating systems, and data base management systems, may directly take over tasks of the D-organization. D-applications run on hardware (computers, networks, etc.). I-applications, such as accounting systems, human resource applications, and Enterprise Resource Planning systems, are information systems in the true sense, i.e., they provide only information for monitoring an enterprise. I-applications typically use the D-applications, similar to the layered integration of the I- and D-organization. Typical B-applications are Decision Support Systems. B-applications commonly use the I-applications.

Regarding coordination (see Fig. 2.6), the physical exchanges can very well be implemented by hardware. Accordingly, the formative exchanges can very well be implemented by digital network services such as e-mail. The informative exchange can be mimicked by using formal languages instead of natural languages, also known as artificial intelligence. For the performative exchanges, one could use a similar 'trick' as for the informative exchanges. However, that is not a good practice since it causes a lot of confusion about responsibilities, data ownership, etc. [Die06, p.123].

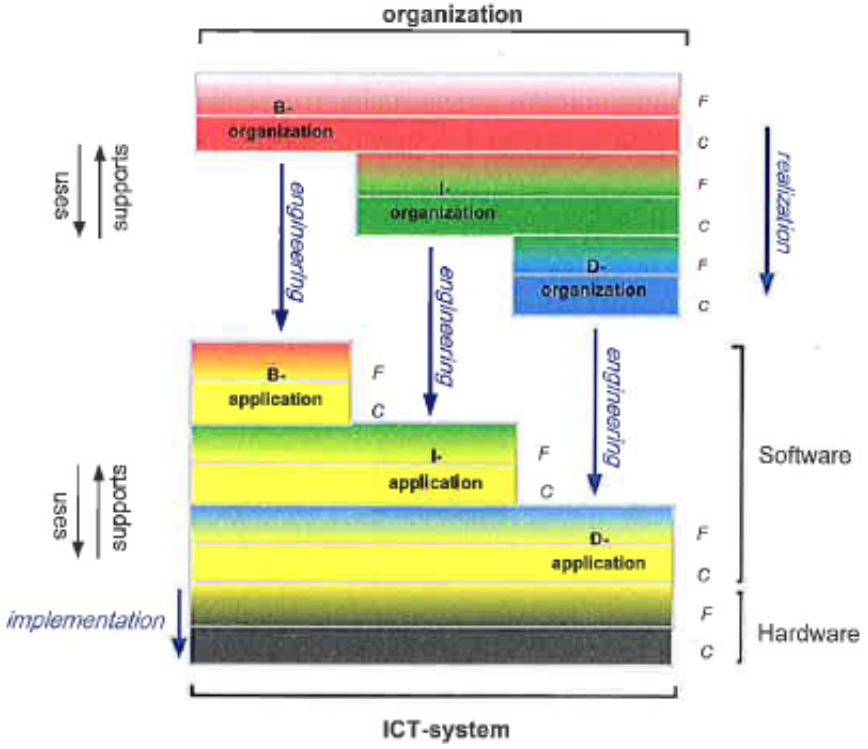


Figure 2.11: Organization and ICT system) [Die08]

Regarding production, a lot of D-applications for the storing, transporting, copying, etc., of (digital) documents are already available. For I-applications, a similar reasoning as for coordination holds: by means of formal languages, I-applications mimick the patterns of the reasoning the human mind applies for deducing or computing new knowledge from existing knowledge. B-applications for production are actually I-applications, only that they are attributed differently than the pure I-organization supporting I-applications because the results of their calculations are taken as real decisions or judgments in the B-organization [San09, p.12]. In principle, the production acts of the B-organization cannot be replaced with applications unless a B-actor is assigned as responsible for the production act.

From the previous paragraphs it must be clear now that actor roles cannot completely be taken over by ICT because of the performative exchanges in which they inevitably participate. So, regarding the implementation of D-actor roles, almost all of their acts can be taken over by ICT, except for the performatice exchanges. In practice, this would mean that one can freely make use of databases and networks but it must always be clear who is eventually responsible [Die06,Die08]. So, each actor roles in the D-organization must be assigned to a human being. A similar reasoning holds for I-actor roles; they must be assigned to subjects in the organization so there is always someone resposible for the functioning of the I-applications, but most of their work can be taken over by ICT. Although Jan Dietz states that B-actor roles cannot be implemented with ICT, and only with human technology [Die06, Die08], in this research it is assumed a similar reasoning as for D- and I-actor roles holds; as long as decisions can be taken automatically based on B-applications, and as long there is someone responsible for the decision, the decision itself can be performed by a B-application as well.

2.3.2 Relation between the GSDP and this research

In a subsidy agency, almost all decisions are regulated by law: for example, if a subsidy application meets some conditions as defined in the law, then it should be granted. Whether a subsidy application meets some conditions, can easily be checked by a computer, provided that the conditions can be formalized and if no human intervention is necessary. Also in other enterprises, decisions sometimes can be taken by computers. If it is necessary the decision is taken by a human being, it suffices for the information system to show the relevant information and buttons for the possible outcomes.

In this research, it is tried to implement the whole B-organization in a Normalized System, positioning the software primarly as B-application. If a decision cannot be formalized or human intervention, for any kind of action, is needed, the information system should accomodate that. As stated in the previous paragraph, there should however always be someone responsible for the decisions that were taken by the information system. Because NS is aimed at creating an *information* system (see Chapter 3), that term will also be used when refering to a computer-based system that in fact takes over a large part of the B-organization.

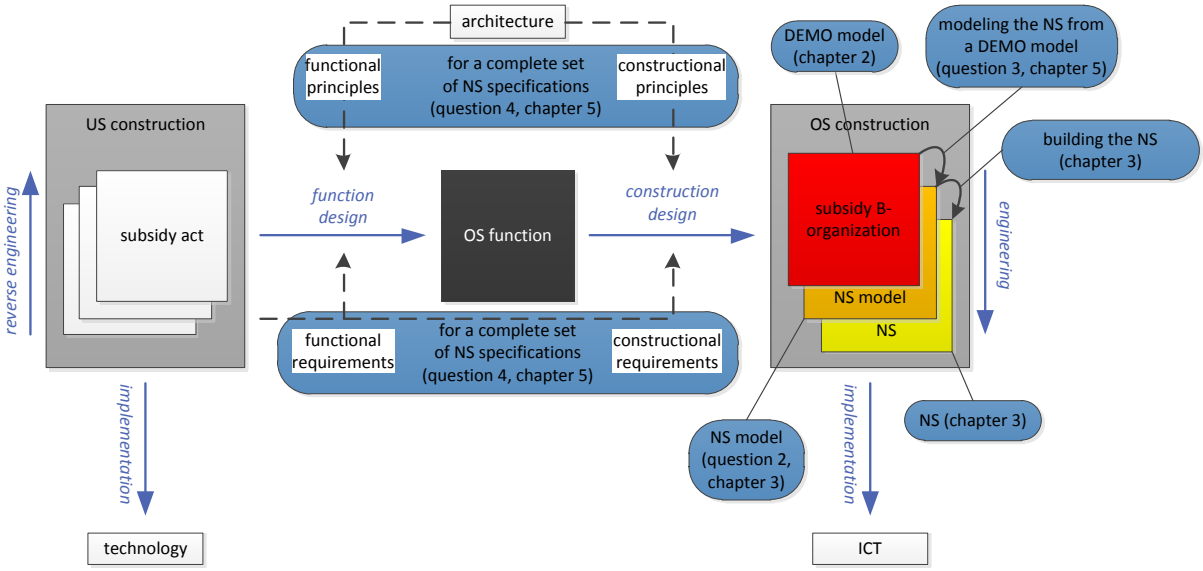


Figure 2.12: Relation between research approach and GSDP

This way of developing a (software) system, is also known as Model-driven Engineering (MDE), although MDE is not tied specifically to ICT. The idea promoted by MDE is to use models at different levels of abstraction for developing systems, thereby raising the level of abstraction in program specification. An increase of automation in program development is reached by using executable model transformations; higher-level models are transformed into lower level models until the model can be made executable using either code generation or model interpretation [dH09].

3. Normalized Systems

The basic assumption of the Normalized Systems approach is that information systems should be evolvable over time and thus should be designed to accomodate change [HBNV10a]. Lehman's law of increasing complexity however states that as an information system evolves over time, its complexity increases unless work is done to maintain or reduce it [Leh80a, Leh80b, LR01]. Therefore, information systems should exhibit stability towards a set of anticipated changes, i.e., the impact of a change is only dependent on the nature of the change itself [MV09, pp.106-107]. If a change requires increasing effort as the information system grows, this is called a *combinatorial effect*. Normalized Systems are defined as information systems exhibiting stability with respect to an anticipated set of changes [MV09] and avoiding the occurence of combinatorial effects, defying Lehman's law of increasing complexity.

The Normalized Systems approach deduces a set of four principles, or design guidelines, to identify and circumvent the problems as mentioned above:

Principle 1. **Separation of concerns** implies that every change driver or concern should be separated from other concerns;

Principle 2. **Data version transparency** implies that data should be communicated in version transparent ways between components;

Principle 3. **Action version transparency** implies that a component can be upgraded without impacting the calling components; and

Principle 4. **Separation of states** implies that actions or steps in a workflow should be separated from each other in time by keeping state after every action or step.

The design principles show that existing software constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner. Normalized Systems therefore proposes to encapsulate software constructs in a set of five higher-level software elements. These elements are modular structures that adhere to the design principles, in order to provide the required stability with respect to the anticipated changes [MV09], and are elaborated in the next section. However, while the latter three principles are completely taken care of by the elements, it is up to the designer to fully adhere to the principle of separation of concerns: "the more fine-grained the identification of the tasks by a designer, the more tasks are separated from each other" [MV09, p.112]. In Section 3.2, a modeling technique for NSs is proposed and in Section 3.3 it is shown how an operational information system is built from such a model.

3.1 NS primitives

The elements as elaborated on below are independent of a specific technology environment. However, the implementation of these elements, must take place in a specific technology environment. The internal structure of the primitives can be described by a design pattern. Every

design pattern is executable, and can be expanded automatically, enabling the automation of the software development process [MV09].

Two kinds of tasks (a set of instructions that performs a certain functionality) are distinguished: a functional task is a task performing a specific functional operation, and a supporting task is a generic task performing cross-cutting concerns.

3.1.1 Data element

A data element represents an encapsulated data construct that contains various attributes or fields, with its get- and set-methods to provide access to their information in a data version transparent way in order to meet the second principle [HBNV10a]. Generic supporting tasks, also called cross-cutting concerns, for instance access control and persistency, should be added to the element in separate constructs [MV09]. A data element is also called a lifecycle object. A lifecycle object knows exactly one lifecycle or (work)flow (see Section 3.1.3).

Mannaert and Verelst even state how the data element or lifecycle objects can be identified [MV09, p.147]: "Data elements (...) basically correspond to the nouns of a business process." They also state there is some room left for interpretation in identyfing the lifecycle objects: "[sometimes the question arises whether] a noun in a business process description is just an attribute of a data element corresponding to a more important data element, or whether it is a lifecycle object of its own with a corresponding flow". However, changing this in a later stage should be no problem since this kind of change can be mapped onto the set of anticipated changes the information system supports.

3.1.2 Action Element

An action element contains a core action representing a specific functional task. Arguments and parameters need to be encapsulated as separate data elements, and cross-cutting concerns like logging and remote access should be added as separate constructs. Four different implementations of an action element are distinguished:

Standard action In a standard action, the actual task is programmed in the action element and performed by the same information system.

Manual action In a manual action, a human act is required to fulfill the task. The user then has to set the state of the life cycle data element through a user interface, after completion of the task.

Bridge action A process step can also require more complex behaviour. A single task in a workflow can be required to take care of other aspects, which are not the concern of that particular flow. A bridge action creates another data element which goes through its own designated flow. At this point it is assumed that any data element that is created by another data element reports its status to the data element it is created. In that way it is possible for the creator to wait for some state of its child(ren) to continue.

External action When an existing, external application is already in use to perform a certain action, the action element can be implemented as an external action. Such an external

action calls another (information) system and sets its end state depending on the external systems' reported answer.

According to Mannaert and Verelst, an action element corresponds to a verb of a business process [MV09, p.147]. Since every action element is dedicated to a specific flow and thus has a specific lifecycle data element, the name should be a concatenation of the name of the data element and the action verb, e.g. SubsidyFormalEvaluator or SubsidyPayer.

3.1.3 Workflow element

Based upon the first and fourth principle, a workflow has to be separated from other actions. These action elements must be isolated by intermediate states, and the information system has to react to these states. A workflow element thus contains a sequence in which a number of action elements should be executed in order to fulfill a workflow. A consequence of the stateful workflow element is that state is required for every instance of use of an action element, and that the state therefore needs to be linked or be part of the instance of the data element serving as argument. This data element is called the life cycle data element of a flow. Since every data element knows exactly one flow, the name of the flow is always the concatenation of the name of the data element and the word "Flow", e.g. SubsidyFlow. It is considered every data element knows such a flow.

3.1.4 Connector element

A connector element represents an input and/or output task of an information system for both user interaction (user connector element) and other applications (protocol connector element). This element is not elaborated on since the focus is on creating an operational information system for an organization while we don't consider graphical user interfaces or connections with existing applications. This however doesn't mean we assume graphical user interfaces are not important for the operation.

3.1.5 Trigger element

A trigger element represents the activation of an action (or workflow) element on a periodic basis. It controls the states and checks whether an action element has to be triggered. This element is not used in this research.

3.2 Modeling a Normalized System

As learned from the previous sections, a Normalized System can be expressed in five primitives, called elements. In this section it is outlined how these primitives can be described. The connector and trigger elements are not explicitly defined in this research and therefore are not elaborated on.

A data element is described by its name and a set of attributes consisting each of a name and a type. Such a type can be another data element or a primitive data type. A primitive data type

is a basic data type provided by a programming language as a basic building block. The actual range of primitive data types that is available depends on the specific programming language that is being used. However, classic basic primitive types include:

- character;
- integer (whole number);
- floating-point number;
- fixed-point number; and
- boolean (true or false).

Some more advanced languages also know String (sequence of characters) and Date as primitive data type. It is considered that data elements should only use these primitive data types so that they can be expanded for any technology environment.

An action element is described by its name and type. In this research, however, this is not considered to be sufficient to eventually build the information system. In order to be technology environment independent, it is not possible to provide actual code in some programming language. Instead, if possible, a description or pseudocode is provided. Pseudocode can later be transformed, perhaps even automatic, into the choosen programming language.

A workflow is described by a set of tripels, consisting of a trigger state (start state), transition action, succes state (end state), and, optional, a failure state in order to allow branching. It can also be represented in a state transition diagram. An example state transition diagram is shown in Fig. 3.1. Circles represent states and rectangles represent actions. Every action is followed by (at least) one state, and every state is followed by (at most) one action.

Both data and action elements can contain cross-cutting concerns, or generic supporting tasks. So, next to identifying the elements, the supporting tasks have to be identified. With this limited set of specifications, as summarized in Table 3.1[1], the builder of the information system is still free to choose any technology environment. The process of building the information system from these specifications is shown in the next section.

Required specification		**Format**
Elements	Data elements	Name, set of attributes (name, type)
	Action elements	Name, type, description/pseudocode
	Workflow elements	Name, state transition diagram
Cross-cutting concerns		Name, description

Table 3.1: Specifications required to build a Normalized System

[1] answering question 2

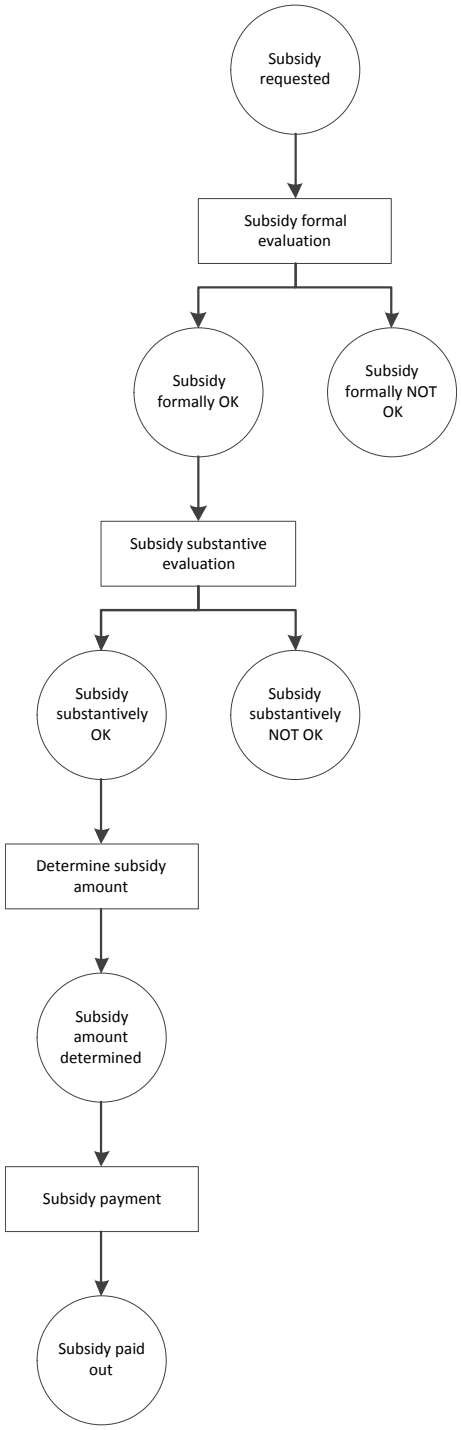


Figure 3.1: State transition diagram for a (governmental) subsidy agency

3.3 Building a Normalized System

When the elements and cross-cutting concerns are defined, the design patterns can be expanded. A so-called pattern expander creates the files for a specific technology environment. One could, for example, build and use an expander to create an information system based on Microsoft .NET.

In Antwerp, an expander is created that generates the code for a set of elements, while taking care of the cross-cutting concerns. An expander creates all the classes and other necessary files that can be deployed onto a web application server. The expander of Antwerp generates various types of source files, among which are Enterprise JavaBeans (EJB) classes, (Apache) Cocoon action classes, Jonas services classes, other Java classes, XML content files, and XSL taglibs. The complete specification of this expander is provided in Appendix F.

Except for expanding the patterns, workflows must be configured, and the (standard) actions and cross-cutting concerns must be programmed. Configuring workflows must be performed manual at the moment but could be automated as well. Cross-cutting concerns are often the same for different applications. Once programmed, they can easily be re-used. The expander of Antwerp already offers the insertion as well as implementation of 10 cross-cutting concerns. And if psuedocode can automatically be turned into real code for any technology environment, then the whole process of building a NS is automated. In Fig. 3.2, for each specification it is shown how it is processed.

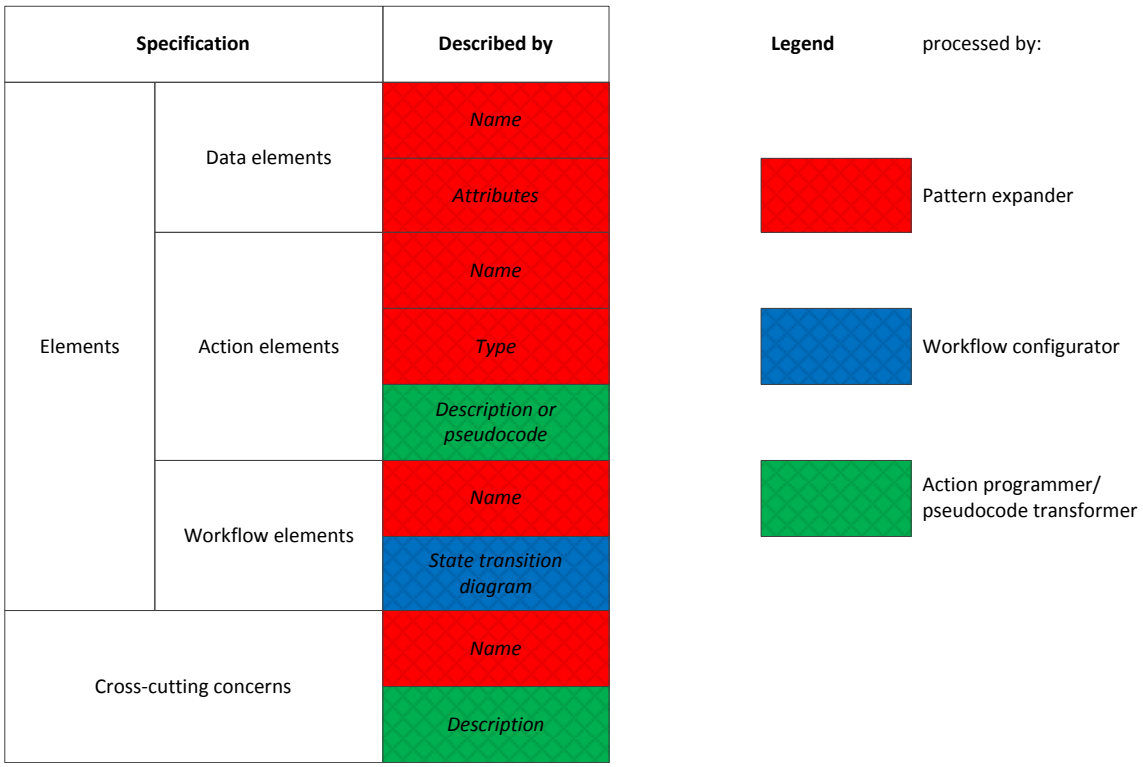


Figure 3.2: Process of building a Normalized System

Part II

Analysis

4. THEORETICAL COMPARISON

When identifying NS specifications from a DEMO model, one has to be sure not to infringe the NS principles. One only has to explicitly take care of the first NS principle, separation of concerns, that states that every task, or action, should be separated; the other principles are taken care of by the implementation of the elements themselves. Since the organization theorem is deduced from the four axioms of the ψ-theory, only the four axioms are tested against the NS principle of separation of concerns.

4.1 Operation axiom

The operation axiom is about actors, fulfilling an actor role, performing coordination and production acts. The discern of different acts adheres to the NS principle of separation of concerns. Subjects can fulfill many actor roles within an organization and in every role it may execute many tasks. However, a subject can only perform an act if he or she is assigned to do so. This is considered a cross-cutting concern and is elaborated on in the next chapter.

4.2 Transaction axiom

The transaction axiom states that coordination acts are performed in universal patterns; the different acts (request, promise, etc.) are clearly separated, adhering the principle of separation of concerns. Another thing learned from the transaction axiom is that sometimes an actor must make a decision. The principle of separation of concerns now tells that the decision making should be split from performing the C-act that follows. This was already found by Huysmans et al. [HBNV10a,HBNV10b]. Van Nuffel et al. [NHBV10] have already shown how to deal with the cancelation patterns. If a C-act is performed explicitly, some action must be performed. If a C-act is performed implicitly (there is some other action that also counts for this one), in fact nothing happens for that particular action. If a C-act is performed tacitly, also nothing must happen. However, in the two latter cases, the state must change so that it is clear it has happened, although no real action was performed. So, in a NS, in fact every C-act is performed explicitly although some actions do not really have another effect than a state change.

4.3 Composition axiom

The composition axiom states that every transaction is enclosed in some other transaction, or is a customer transaction, or is a self-activation transaction. So, a transaction is either enclosed in another transaction, or is a root transaction, either self-initiated or initiated by an (external) customer: a transaction that is not enclosed in another transaction and in which other transactions might be enclosed. Regarding the enclosing of transactions, attention must be paid in order not to infringe the principle of separation of concerns:

It is possible that one actor starts multiple transaction of the same kind (for different entities). Examples are found in the library case from Dietz [Die06, p.218] as well as in the subsidy agency case (Chapter 7). Sometimes, the actor that creates the transaction instances, has to perform an act for each transaction that has been completed. For example, for all subsidies that have been inspected, it has to be checked whether the inspection result differs from the original application.

The checking of each individual inspection result clearly is not part of the overall flow in which all the inspections are initiated. However, they can not be part of the inspection flow either; if it was part of the inspection flow, it would mean that this check is always performed after inspection while other actors could start an inspection for other reasons as well. So, the handling of each individual inspection, is part of yet another, actor specific, flow.

Attention must also be paid regarding passing information passing actors: in general, when an actor deals with an agendum, it performs a C-act. The C-act is addressed to another actor (role) and the resulting C-fact is placed on the addressee's agenda. However, wait conditions between enclosed transactions can be present such that one actor (role) has to wait for a C-fact while he doesn't participate in the performative exchange in which that C-fact is created. Unfortunately, such an example is not present in the case subsidy schemes, but it is in Dietz's pizza case (Appendix C) in which the deliverer (A04) of a pizza purchase has to wait with delivery until preparation has been completed (transaction T02 with participating actor roles A01 and A02). From Dietz [Die06] it was learned that an actor knows about such a fact simply because it is on his agenda. However, for an implemented pizzeria, the C-fact T02/ac (completion of preparation) somehow must be available to A04 (deliverer).

Making information available to the actors, is in fact the concern of the I-organization. To move information between two subjects, two strategies are known [SLSLW03,MF98,Boy07]: the push strategy in which information is pushed from one actor two another, and the pull strategy in which one actor pulls information from the other one. For the pizza case, this means either A01 or A02 must tell A04 when the preparation has been completed, or A04 must (frequently) check whether the preparation has been completed.

All actor roles in a transaction composition can be placed in a family tree: the actor roles become the vertices, the transactions the edges between them. If an actor (role) initiates some transaction, the executor of that transaction becomes a child of the initiator, see Fig. 4.1. It should be clear now that an actor (role) always knows its parent (unless it is the root and has no parent) and its children (if it has any), because they are the other party (either as initiator or as executor) in any transaction the actor (role) is involved in (either as initiator or as executor). Now the principle of separation of concerns tells that an actor should not know about its sibling(s), grandchild(ren), grandparent, etc., simply because it is not his concern; an actor should only know its parent and/or child(ren), it should not know it is part of a bigger whole nor should it know about the structure of the rest of the enterprise.

So, for the pizza case this means A04 cannot wait for T02 to be completed, because it does not know about the existence of T02. Instead, he can only wait, as part of the agreement between A01 and A04[1], until he is told to continue his duties for a particular purchase. This also rules out the pull strategy for information passing between actors, which is considered a bad practice in software design anyway because it is processor consuming [HR06]; the pull strategy requires a frequent activity while pushing information only requires one activity. The other way around, A02 does not know about A04 and thus cannot inform A04 about the completion

[1] In large enterprises known as SLA

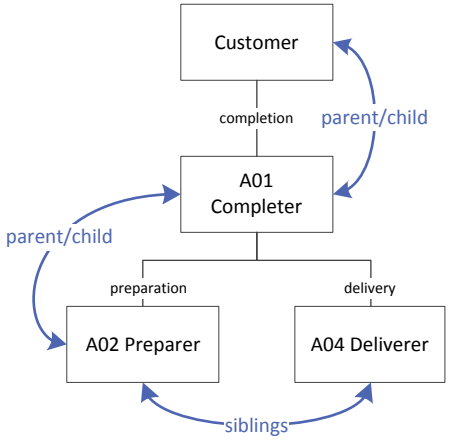


Figure 4.1: Family tree of the pizzeria actors

of T02. However, A01 knows both T02 and A04 and thus can tell A04 when T02 has been completed.

So, following the principle of separation of concerns, information may only be passed between two actor roles who are commited in a transaction. If information must be passed between actors that are not directly related, extra actions will be needed to accomodate that. In general, information that is required for wait conditions, should be passed (pushed) through the common anchestor. It must be noted that with the DEMO-2 style of a PM it is easier to detect these kind of situations as DEMO-2 clearly shows the actor roles that are involved in a wait condition. Because such a situation is not present in the subsidy agency, the implementation is not investigated in this research.

4.4 Distinction axiom

The distinction axiom states that there are three distinct abilities, namely performa, informa, and forma. This helps in identifying different actions for a single P- or C-act. For example, a C-act consists of many communication acts at different levels (Fig. 2.6). Applying the principle of seperation of concerns also tells it is better to separate the P- and C-acts from the flow in which it resides. In the overall flow these acts simply become an action of type Bridge, creating a lifecycle object that performs the P- or C-act in its own flow. The latter flow consists of different actions that can be distinguished by applying the distinction axiom. In this way, coordination 'channels' can be re-used, forming true building blocks.

4.5 Concluding remarks

From this chapter, it was learned that some care must be taken when identifying NS elements from a DEMO model. One has to be sure not to infringe the NS principle of separation of concerns. Most axioms of the ψ-theory, underlying DEMO, support or acknowledge this principle[2]. However, the composition axiom allows room for violating the principle of separation

[2]answering question 1

of concerns: if a component structure of the involved actors is viewed as a family tree, a component (actor) only knows its parent (if it has one, otherwise it is the root) and its children (if it has one or more) while it does not know its sibling(s), uncle(s), etc. If information must be passed between any two components, it must be done through the (lowest) common anchestor. It was also learned that if multiple instantiations of a transaction kind are started, the handling of each transaction must be split of from the transaction that is initiated and from the transaction from which it is initiated, it is another concern (Fig. B.2). It would be superfluous to split of single instantation of a transaction kind, as it would introduce more flows that only consist of the starting of another flow (Fig. B.1). From the other actioms, it was learned that decision actions must be split from P- and C-acts, that the distinction axiom can help in identifying different actions, and that P- and C-acts may become a lifecycle object of their own, and thus form true building blocks.

5. IDENTIFYING NS SPECIFICATIONS

In Section 5.1, the NS specifications, being the cross-cutting concerns and the elements, are tried to be identified from a DEMO model. Because a DEMO model leaves out implementation details, it is likely not all specifications can be fully defined from a DEMO model. In Section 5.2, it is identified whether the set of specifications is complete.

5.1 Identifying NS specifications from a DEMO model

In the previous chapter, it was learned some care should be taken when identifying the NS element from a DEMO model. Now it is time to find how exactly a NS can be defined from a DEMO model. It is tried to discern the specifications for a NS in the different artefacts of a DEMO model. From the ψ-theoryand the DEMO models, eight DEMO artefacts are distinguished [Wan09][1]:

- Object class[2];
- Scale dimension;
- Fact kind;
- actor role;
- Transaction kind;
- Transaction step (both C- and P-acts);
- Information bank (containing C- and/or P-facts); and
- Action rule.

In the next sections, it is investigated which NS elements and cross-cutting concerns can be identified from these artefacts.

5.1.1 Object class, scale dimension, and fact kind

As learned from a previous study [Kro], DEMO entities are similar to NS lifecycle objects. It was also stated there that every object class from a DEMO model can become a data element with a flow passing the three states, i.e., prenatal, operational, and postmortem (Fig. 5.1).

A scale dimension can also become a data element, enabling re-use. The scale dimension is e.g. time, money, length, or temperature. The exact scale (hh:mm:ss, D-M-Y, dollar, euro, meter, kilometer, °C, °F, ...) is not mentioned in a DEMO FM because on the ontological level

[1] The artefact 'transaction phase' is left out as it does not add anything in our point of view.
[2] A category is a kind of object class and is therefore not mentioned separately.

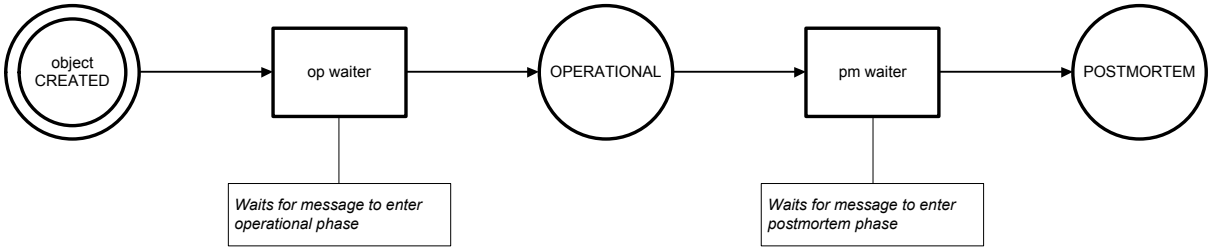


Figure 5.1: Workflow for a data element representing an object class

it doesn't matter. In software, however, the scale dimensions must be represented by set of primitive data types (see Section 3.2).

Scales are always external and in that way similar to external object classes or categories. External object classes or categories also have to be represented by a set of internal types.

Binary fact kinds are like relations between objects[3]; since an attribute of a data element can also be a link to another data element, a binary fact kind can be represented by attributes of data elements.

Calculating a derived fact of an object class is considered an I-transaction, devised in the realization of the organization (Section 2.1.5). However, with derived facts, there is always a problem of when and how to calculate it: one could calculate a derived when needed, but this could mean the same calculation is performed on the same data over and over again. One could also store the result of such a calculation but that means it has to be synchronized if the original data changes. Both approaches have their advantages and disadvantages, also depending on the situation.

So, it can not determined (yet) for all organizations how to model the calculation of derived facts in a NS. The derivation rule could be programmed in an action element but other solutions might be better. It is however assumed these derived attributes are available as a normal attribute. This also suggests that the availabity of derived facts is a cross-cutting concern, a supporting action that allows a data element to use its derived facts, based on original facts.

5.1.2 Actor role

Actors and actor roles do not exist in an information system as such. The information system, or its engine, fulfills all actor roles it gets assigned, i.e., all the actor roles the enterprise whishes to be automated. Since (a subject fulfilling) an actor role can perform many actions, it is possible the information system only takes over a part of the set of actions. If a subject fulfills an actor role for some step regarding a certain object, it is likely that subject should also fulfill that same actor role in another step, e.g. if some person in an organization takes the request for a subsidy grant, he or she should also inform the subsidy applicant whether the subsidy is granted or not. However, an organization can choose to arrange otherwise. Moreover, in implementing an organization, subjects are assigned to fulfill one or more actor roles and authority may be delegated. This should all be handled in a cross-cutting concern authentication for action elements that handles who may perform which process step(s) for which instance of

[3]Unary fact kinds are object classes

an object class. There should also be a cross-cutting concern authorization for data elements that takes care only the right people can access data.

5.1.3 Transaction kind

From the theory (Section 2.1.2) it was learned that a transaction takes place in a standard pattern, may be enclosed in another, and has a result that concerns one or more objects. Because different transactions can act simultaneously on the same object, known as parallel execution, all transactions should have their own workflow. So, a transaction is a lifecycle object or data element with links to the object class(es) its result is about and a workflow for passing the different transaction steps (see next section). Creating a life cycle object for every transaction also enhances the principle of separation of concerns: every transaction clearly has just one concern. Regarding the enclosing of a transaction, some notes were already made in Section 4.3.

5.1.4 Transaction step

As learned from the previous section, for every transaction kind, a data element has to be created, every transaction instance being an instance of the matching data element. From Section 4.2, it was learned the different C- and P-acts as well as the decision acts in the transaction pattern have to become action element associated to the data element representing the transaction kind. Identified actions are: request, promise, execute, state, accept, decline, reject, quit, stop, and the four decision-actions validateRequest, validateState, validateDecline, and validateReject. Two more actions have to be added: re-request and re-state, because in practice they will differ from the initial request and state action.

However, if all these actions are put on the same data element representing a transaction, what does it mean to perform a request action? The answer lies in the fact that coordination is between two actor roles, or actors: the initiator and the executor of the transaction.

So, a 'transaction element' is split into two elements: one for the initiator and one for the executor. For each transaction a TransactionInit and a TransactionExec data element are created, each with a flow containing the steps that are performed by the initiator and the executor of the transaction respectively. The execution part is created when the request of the init part is performed; it is not until then the executor knows about the transaction. Wait actions are introduced when an actor (role) has to wait for a C-act from the other involved actor (role). In general, the initiator does not know whether the executor will promise or decline the request. So, the wait action is not for just a promise or just a decline, but for either a promise or a decline. A similar reasoning holds for the wait action of the executor for the accept or reject. The workflows representing the basic transaction pattern are shown in Fig. B.3, the workflows for the complete transaction pattern are shown in Fig. B.4. Every transaction in an organization knows the actions and flows for the complete transaction pattern. However, not every action will be used in an organization, e.g., some transactions are never declined; every transaction uses at least the actions for the basic pattern. C-acts are shown by a dotted line.

Now it can also be outlined how to cope with actor roles that reside outside the organization. Regarding P-acts that are performed by external actors, an enterprise cannot fix how it should be performed. Regarding C-acts, an enterprise can ask external actors to communicate in some

format and before some deadline, it however can only wait for the receival of a C-act and not determine how and when the coordination will be performed. So, acts that are performed by external actors should not be modeled in the enterprises' information system. Because a transaction is split in two parts, one for the initiator and one for the executor, it is clear that if either the initiator or the executor resides outside the organization, that part of the transaction does not have to be modeled. Interesting to see now is that it is still not fixed how the request is made by an external actor: for example, whether the external actor can enter it through a website, or whether it is a desk employee that enters the data on behalf of an external actor that arrived at the desk.

Although many actor roles may initiate some transaction, it is now also clear that is not wrong to model just one (internal) initiator: it is assumed that all internal initiators use the same guidelines for acceptance etc.

5.1.5 Information bank

Internal information banks can be seen as a log containing all C- and P-facts of the transaction kind the bank represents. In order to have such a log, or information bank, all facts must be saved. Remembering and storing a fact is the an I- and D-transaction respectively [dJ09]. However, since these transactions should be there for all transaction kinds, this is a cross-cutting concern for all data elements representing a transaction.

Information banks also contain instances of object classes. A cross-cutting concern persistency should make sure data about objects is saved. Again, remembering and storing these facts is actually a concern of the I- and D-organization. However, now it is clear the storing should be taken care of for all facts in the same way, and not for each fact individually in its own way.

If information must be retrieved from an external information bank, it is often captured in a derivation rule, based on one single attribute that is stored for external object classes. Again, for derivation rules it is not yet clear how to model them in a Normalized System.

5.1.6 Action rule

The action rules can be read from the ARS. In Section 4.3, it was stated that some extra actions might be needed in order not to infringe the NS principle of separation of concerns. In Section 5.1.4, it was stated that C- and P-acts are separated from the actual decision-making.

From the basic action rule syntax (Listing A.1), it can be read that the first line mentions the start state of a transition. The second line is the decision action which brings a flow either in a succes state or in a fail state, e.g., PaymentAcceptable, or PaymentNotAcceptable. The third and fourth line tell what action should follow from either state. So, from the ARS, pseudocode or a description of the decision action, and the transition from a start state to a succes and fail state can be read.

5.2 Completeness of specifications

In the previous section, it was shown that a NS needs to provide three cross-cutting concerns: logging and persistency (for creating the information banks) and authorization (for making sure only an authorized person is able to perform a particular transaction step). It was also shown which elements can be defined from the different DEMO artefacts. In this section it is checked whether the elements can be fully defined.

5.2.1 Data elements

Data elements are created for DEMO object classes and scale dimensions. For internal object classes it is clear what they are and how they are represented. External object classes and scale dimensions must be represented by primitive data types. This information is not available in the DEMO model. Data elements are also created for the transaction kinds: two per transaction kind. The data element representing a transaction kind holds links to the data element(s) representing the object class(es) the result of the transaction kinds is about.

5.2.2 Action elements

Action elements are defined from the different transaction steps, which are in general the same for each transaction kind. The implementation however will in general differ per transaction kind and therefore this is not considered to be a cross-cutting concern. Altough the names and descriptions can be read from the DEMO model, the type cannot be found there.

In Section 4.4, it was argued that C- and P-acts often become a data element of their own with a designated workflow in which different actions are performed to bring about the C- or P-fact. The action that starts the C- or P-act then is of type Bridge. If the action is to be performed manually, the action must be of type Manual. Only in rare cases it will be of type Standard, performing a single action that is never re-used. Regarding decision acts, a similar reasoning holds except for that it will always be of type Standard or Manual. It was already learnt that decision rules can be found in the ARS. If a rule can be formalized, it can become a standard action. Otherwise, it must be performed by a human being (manual action). So, for each C-, P-, and decision act, it must be known whether it is performed manual or automatic.

5.2.3 Workflow elements

Every object class knows a workflow as was shown in Section 5.1.1 and every transaction knows a workflow as was shown in Section 5.1.4. The workflow of the latter one can be adapted because of wait conditions and the enclosing of other transactions. However, all information for describing a workflow element with its state transition diagram is present in a DEMO model.

5.3 Concluding remarks

In this chapter, the cross-cutting concerns and elements were identified from a DEMO model. Three cross-cutting concerns were found that a NS needs to exhibit when being defined from a DEMO model: logging (for action elements), persistency (for data elements) and authorization (for both data and action elemets). It was also shown how NS elements can be defined from different artefacts in a DEMO model. All these artefacts are present in the AM since the AM is the most detailed and comprehensive aspect model and contains all information that is also present in the other models (Section 2.2.3). However, not all information can easily be read from the AM. It is easier to find some of the artefacts in other models as well: the object classes can be read from the FM, the transaction kinds from the CM or TRT, while only the AM contains the action rules. The number of enclosed transactions can be read from the PM[4].

It was found that data elements can be defined from the object classes and transaction kinds. Action elements are in general the same for each transaction kind, however will differ in implementation. In some cases, extra action elements must be defined to deal with wait conditions in order not to violate the principle of separation of concerns. Every data element knows a flow, either a transaction flow (configured with information from the ARS) or an entity flow (the same for all object classes).

To describe the elements completely, extra information is required[5], namely:

- For each C-, P-, and decision act, whether it is performed automatic or manual; and
- For each of the external object classes and dimensions scales how it is represented internally by primitive types.

It is stated once more that it doesn't matter if not all elements are identified. Because of the ability of the information system to change, it is easy to add extra elements in a later stage.

[4] answering question 3
[5] answering question 4

Part III

Design and validation

6. DESIGN

In the previous chapters, it was learned what specifications are needed to build a NS, namely cross-cutting concerns and elements. It was learned how to identify the elements in the DEMO artefacts. Moreover, three cross-cutting concerns were identified. In this chapter, the method to come from a DEMO model, supplemented with some extra information, to a description of a NS which can be turned into actual code that can be deployed on a web server or alike, is presented.

6.1 Cross-cutting concerns

In the previous chapters, three cross-cutting concerns were identified that must be present for every NS that is created from a DEMO model. First, for all actions representing a C- or P-act, logging must be performed. Second, all objects must be stored in a database. This is handled by a cross-cutting concern persistency. Third, there should be a cross-cutting concern that makes sure only authorized persons can perform some action. The ABD can be used to make sure actors can access the data they need to perform actions. Other cross-cutting concerns may be defined for an organization as well, e.g. remote access, but are not gathered from a DEMO model.

6.2 Elements

The elements have to be identified for each organization indivually. The information that is needed to do so, is the following:

1. A DEMO model of the B-organization, consisting of all four aspect models: a CM (both ATD and TRT), PM, FM (both SSD and FDL), and AM;
2. For each kind of dimension and external object class: how it is represented by a primitive data type (Section 3.2);
3. For each C- and P-act: whether it is performed automatic or manual; and
4. For each decision (ARS): whether it is performed automatic or manual.

In the next sections, the NS elements are identified from the different DEMO aspect models, supplemented with some additional information. The order as presented turned out to be the best order to perform the steps. One could however adopt another order if desired.

6.2.1 Fact Model

As learned from Section 5.1.1, for each scale dimension and external object class or category, a data element has to be created. Internally, it is represented by one or more primitive type(s),

as described by item 2 from the list above. Also, for each internal object class or category a data element is created with a workflow (element) as exhibited in Fig. 5.1. Its attributes are links to other data elements representing object classes as well as links to the data elements representing a scale dimension, mimicking the reference laws between them.

For derivation rules from the FDL, it not yet known how to model them. It is considered the results are available as attributes and therefore they must also be assigned as an attribute.

6.2.2 Construction Model

From the CM, or the TRT, the transaction kinds can be read. As learned from Section 5.1.3, for every transaction kind, two data elements are created. However, is the transaction is initiated or executed by an external actor only, only one data element is created. Every data element representing a transaction contains links to the data elements describing the object classes the transaction result is about, and knows at least the basic actions and flow as exhibited in Fig. B.3. The further identification of actions is performed using the AM (Section 6.2.4), the type of the actions are provided by items 3 and 4 of the list above.

6.2.3 Process Model

From the PM it can be read whether multiple instances of a transaction kind are initiated by some actor role from another transaction executed by that same actor role. If that is the case, than it is often the case that that actor role handles each initiated transaction kind in a similar way. From Section 4.3 it was learned that the handling must be split of from either of the transactions as is shown in Fig. B.2. If only one instance of a transaction kind is initiated, this extra handler is not needed.

6.2.4 Action Model

The ARS is used to identify action elements that are not part of the basic pattern and might be part of the complete pattern (Fig. B.4). It also tells how the workflows of the transaction elements must be defined. An action rule is provided in a basic syntax: the first line states the start state, from the second line, a decision action can be defined, and the third and fourth line state the follow-up states and actions. If a decision can be taken automatically (stated in item 4 of the list above), the formalized rule can be used as pseudocode. If another transaction kind is initiated (starting an enclosed transaction), an action of type Bridge must be inserted (see Fig. B.1). The use of the AM in identifying action elements and defining workflows, is further shown in Chapter 7.

7. VALIDATION

In this chapter, it is shown how the gained method from the previous chapter is applied in practice. Only the part to define the elements is shown, the cross-cutting concerns are the same for each case (see Section 6.1). A case from Capgemini, called Governmental Subsidy Schemes, is used for demonstration and will be introduced shortly.

7.1 Governmental Subsidy Schemes

A subsidy, according to the Algemene Wet Bestuursrecht (ADW) (in English: General Administrative Law Act) [Rij, art.4:21.1], is an entitlement to financial resources provided by an administrative authority for the purposes of specifically named activities of the applicant, other than by way of payment for goods or services supplied to the administrative authority. For all kinds of activities, different subsidy schemes exist. Getting a subsidy is divided in two phases:

Before performing some activity, a subsidy applicant has to ask for a grant (T01 subsidy grant) in which the applicant states for what activities the subsidy will be used. The agency checks whether all required information is provided (T02 formal evaluation) and checks whether the application satisfies the conditions, or whether the activity is worth a subsidy (T03 substantive evaluation)[1]. If the application surpasses both evaluations, the agency will ask the applicant to really perform the activities (T04 obligations compliance) after which the subsidy is granted, i.e., it is only *asked* to perform the activities (T04/rq,pm) but the activities do not have to be executed yet (T04/ex,st).

After the applicant has executed the activities, he or she asks for final subsidy amount determination (T05 final subsidy amount determination). In general, the agency will provide the subsidy amount as was stated at the grant, but the amount can differ based on the activities that were really performed (T04). When the amount is determined, it will be paid out (T06 subsidy payment).

In some cases, it is not necassary for a subsidy applicant to ask for a grant before final amount determination. That is often the case for activities that require a low subsidy and will be executed even if the subsidy is not granted. In such cases, the agency will ask for a grant on behalf of the subsidy applicant before amount determination.

In a random check (T07 random check), some applications that match a risk profile are inspected. A risk profile determines which types of applications are more unplausible than others, e.g. subsidy for a company car, or for a particular car seller. Unplausible applications are often a way of fraud and lead to unintended payments. Through inspection (T08 inspection), the data of an application are verified. If the data gathered by the inspector differs from the application, the grant of that subsidy will be cancelled and a new one will be started.

In practice, if a canceled subsidy is already paid out, it will not be recovered immediately. Instead, it is balanced with the subsidy amount of the new application. The cancellation of

[1] The splitting of the formal and substantive evaluation actually is not derived from the law but from the existing implementation.

a subsidy grant is accomodated by the cancellation patterns as provided by the transaction axiom (Section 2.1.2). That axiom states a transaction could be rolled back by a counter act but does not tell exactly how to do so. It is assumed, the cancelled action should be neutralized, after which a new action can be started. So, it is not considered a good practice to leave it and balance it with later results. Moreover, this practice is also against the NS principle of separation of concerns: it is not the concern when paying out one subsidy to deduct another one that has been cancelled. Although clients probably will not like it when they have to repay a subsidy and then later get the money back for the renewed application, for now this is considered the best way of solving it. So, recovery (T09 subsidy recovery) will be started if a paid out subsidy is cancelled. The recovery itself consists of a repayment (T10 subsidy repayment) from the subsidy applicant.

Several subsidy schemes were studied at Capgemini, from which only the sloopregeling will be shown. The sloopregeling subsidy scheme aims to contribute to improving air quality in the Netherlands. It does so by providing a subsidy to someone who replaces an old environmentally unfriendly car or van by a car with lower emissions of environmental pollutants. In practice, it is the car seller who applies for the subsidy while he subtracts the subsidy amount of the customer bill. The complete act of this scheme can be found on the website for this scheme [MvV09].

A current implementation with people and ICT exists. When creating a NS, the entire system is redesigned and (thus) reengineered (see Fig. 2.12).

7.2 Determining the input values

The input needed to define the elements, was stated in Section 6.2. These will be gathered first:

1. The DEMO models for the sloopregeling are provided in Appendix D.

2. Random checks are performed every month, so a period is represented by a month and a year. Vehicles in the Netherlands are registered at the Rijksdienst Wegverkeer (RDW) (in English: Governmental Road Services). All vehicle data, such as type, owner, and color, can be required at the RDW from the vehicle's number plate. A person in the Netherlands can uniquely be identified by its Burger Service Nummer (BSN) (in English: Social Security Number). Car sellers, among other companies, are registered at the Kamer van Koophandel (KvK) (in English: trade register) and can be identified by its KvK number. Purchase agreements must be sent along with the application. They can be represented by a link to the place in the file system (either a place in a file cabinet or, when scanned, some place on a hard disk). A verdict is a yes or no, and can be represented by a boolean. Money is represented by a floating point number. It is considered there is an internal type date to represent dates.

3. C-acts between two internal actors are performed explicitly and automatically, i.e., the information system takes care of the information passing between flows as exhibited in Fig. B.4. From the global ATD (Fig. D.1), it can be seen there are four transactions in which C-acts with an external actor are performed: People can apply for a subsidy grant (T01/rq) and final amount determination (T05/rq) through the website of the ministry. Notifications about the granting (T01/st) and the final amount determination (T05/st)

will be sent by mail. T01/pm and T05/pm are performed tacitly and are thus not implemented (empty action). For the sloopregeling, the obligations compliance (T04) is fully regulated by law and therefore T04/rq and T04/pm are performed tacitly. This also means the agency does not have to wait for an explicit T04/pm. The T05/rq counts as the T04/st (T04 implicitly) and the T04/ac again is performed tacitly.

A decision about granting and amount determination can be contested within 30 days. It is considered that if the subsidy applicant hasn't responded within 30 days after the notification has been sent (T01/st or T05/st), the transaction is accepted. A request for re-payment (T08/rq) will be sent by mail as well as the confirmation of having received the re-payment (T08/ac). P-acts are performed automatic for T01, T02, T03, T05, T06, T07, and T09. The exact procedure for P-acts however is not present in the DEMO model. Code is derived from the law and is provided in Appendix E. T08/ex is performed manual.

4. If a decision can be formalized, it will be performed automatic, meeting the principle of the Dutch e-government architecture to execute tasks automatically if possible [eK07, principle 6.1.1.1].

7.3 Defining the elements

A summary of the identification of the elements is stated below. The formal definition of all elements can be found in Appendix E. The re-usable components for sending mail and transfering money are presented in the last section.

7.3.1 Fact Model

From the SSD (Fig. D.4), data elements are created for all object classes and dimensions: Subsidy, FormallyEvaluatedSubsidy, SubstantivelyEvaluatedSubsidy, GrantedSubsidy, FinalAmountDeterminedSubsidy, PaidOutSubsidy, RepaidSubsidy, RecoveredSubsiy, InspectedSubsidy, Period, Vehicle, Person, PurchaseAgreement, CarSeller, Verdict, and Money. For the data element Subsidy a workflow element SubsidyFlow is created with the workflow as shown in Fig. 5.1. Also, action elements are created: SubsidyOpWaiter and SubsidyPmWaiter, both of type External. It is assumed external data about vehicles, person, etc., is available somehow.

7.3.2 Construction Model

From the TRT (Table D.1), the transaction kinds can be read, 10 in total. For each transaction kind, in general, two data elements are created. However, T04 and T10 are executed externally and T05 is initiated externally, so for those transactions, only one data element is created. Note that T01 can be initiated by both an external and an internal actor, so two data elements are needed. In total, 17 data elements are created. Each data element for a transaction holds a link to the data element representing the object class it works on and knows a flow as exhibited in Fig. B.3. The actions for each transaction element and its flow will be further defined with the AM (next section) until at most the complete pattern as shown in Fig. B.4.

7.3.3 Process Model

From the PSD of the random check it can be read that multiple inspections are started from the same random check. So, a RandomCheckExecInspectionHandler is created that handles the initiation of a T08 and the handling when it has finished as determined by the AM.

7.3.4 Action Model

From the ARS, the action and workflow elements can be defined. actor role A01 is the executor of T01 and an initiator of T02, T03, and T04. So, from the action rules for A01 (Listing D.1), the actions for T01Exec, T02Init, T03Init, and T04Init can be defined: from line 2, it can be read a T02 has to be started from a T01Exec, so an action T01ExecT02Creator is defined. The action in line 5, T02InitAccept, is already present in the basic workflow. From line 8, a decision action is defined: T01ExecCheckFormalEvaluationResult. From line 9, a T03Creator is defined and from line 10 it can be read that T01 needs a Decline action. Line 16 shows there is a need for an action T01ExecCheckSubstantiveEvaluationResult and from line 21 a T01ExecT04Creator is defined. The other actions as defined in the action rules for A01, are already part of the basic pattern flow.

For the action rules for the other actors, the identification of actions is similar. Below, the actions that are not part of the basic pattern flow are mentioned:

- Listing D.4, line

  2: T05ExecCheckGrantHasBeenRequested
  3: T05ExecCheckGrantHasBeenGranted
  5: T05ExecCheckGrantHasBeenDeclined;
  6: T05ExecDecline;
  7: T05ExecT01Creator;
  25: T05ExecT06Creator.

- Listing D.6, line

  7: T07ExecT08Creator;
  10: T08InitCheckInspection;
  12: T08InitReject;
  15: T07ExecT08HandlerCheckInspectionResult;
  16: T07ExecT08HandlerT01Cancel;
  17: T07ExecT08HandlerT01Creator.

- Listing D.8, line

  5: T09ExecT10Creator;
  8: T10InitCheckRepayment;
  10: T10InitReject.

The workflows (state transition diagrams) can be found in Appendix E.

7.3.5 Re-usable components

For sending mails, used for T01/st, T05/st, T01/dc, T05/dc, T08/rq, and T08/ac, a MailSender is needed. A MailSender is a data element with a workflow MailSenderFlow that encompasses three actions: MailSenderCompose, MailSenderPrint, and MailSenderSend. The first action composes a message, this can be done automatically with predefined messages. The second action is to print the letter, this is also done automatically. The third one is a manual action in which the letter is sent out. As a letter leaves the agency, or it is dropped in a box for outgoing mail, it is considered to be sent.

The actions for composing and printing are not elaborated on now, they are shown only as example. Printing is a concern of its own, and thus should again be split from the MailSender by a Bridge action. Note that it is also easy to change this mail sender to the modern email sender: the MailSenderPrint actions can now be left out and the MailSenderSend becomes a standard or bridge action to send an email. Moreover, different versions can be used next to each other, enabling the use of different communication channels for different customers.

For the transfer of money, a reusable component Payer is used which only knows one action for transfering the money using some bank application.

7.4 Adding a second scheme

Within Capgemini, several schemes were studied and for each scheme a DEMO model was created. The DEMO models were similar to a large extent: in fact, the CM and PM showed great resemblance while the AM and FM showed some more differences. In terms of NS elements, this means the data elements are to more or less the same. Only new object type, related to a subsidy (what the subsidy is about, i.e., whether it is about cars, glass, etc.) in the FM introduce a new data element. Regarding the action and workflow elements, these are also more or less the same; only the implementation of some of the actions (in particular the decision actions) will differ. This could easily be accomplished by using a new version of these actions and of the subsidy data element.

Part IV

Conclusions and recommendations

8. Conclusion and discussion

This chapter concludes the entire thesis project and discusses some relevant issues for future work.

8.1 Summary

The goal of this thesis was to find a method to come from a DEMO model to a Normalized System. DEMO is a methodology for the design and engineering of organizations. DEMO abstracts from the implementation of organizations and is therefore considered a good starting point for defining a supporting information system. A Normalized System is an agile information system, free of combinatorial effects and able to adapt easily to changing needs over time. This is achieved by adhering four principles and by the use of five basic elements. A Normalized System can be modeled by describing its elements and cross-cutting concerns (generic supporting tasks) from which it can be built automatically using a so-called expander.

Composing the method was achieved in a few steps. First, the theories were compared, verifying whether the DEMO theory matches the NS principles. It was learned that that was the case but also that some care must be taken with the enclosing of transactions. It was also learned that modelling an organization in DEMO, helps in adhering to the glsns principles.

The different artefacts of a DEMO model were transformed into NS elements. Also, three cross-cutting concerns were found that the NS need to exhibit: persistency, logging, and authorization. Moreover, it was checked whether these specifications were complete. It was found that with just a limited amount of extra information, as specialization of the DEMO model, the NS could be fully defined. However, it was also found that for derived facts from a DEMO model, it is not yet clear how to model them in a Normalized System. The found method was illustrated with the case Rijkssubsidieregelingen from Capgemini.

8.2 Contributions

For both DEMO and NS it is interesting to see that both theories adhere to the idea that seperation of tasks is crucial to fully define an organization and information system respectively while the first theory is focused on the business while the second one is focused on information systems. In DEMO this is represented by the fact that a transaction can enclose other transaction, separating the different 'sub'tasks. In DEMO, the separation of tasks is also recognized in the distinction axiom which distinguishes three different kinds of actions, i.e., ontological, infological, and datalogical. In NS, the separation of tasks is established in a principle and implemented in an action element.

For DEMO, it was found that I- and D-transactions are mostly generic supporting tasks which could be used by all B-transactions. This seems to contradict with the theory of de Jong and Dietz who showed the identification of I- and D-transactions *per* B-transaction. The B-transactions are considered to be the unique functionality that make business, while the I-

and D-transactions only support the B-transactions. The NS principles clearly distinguishes between unique functional tasks and generic supporting tasks and it is clear that I- and D-transaction are such generic supporting tasks while the B-transactions are the unique business functionality.

For NS, DEMO showed to be a nice way for identifying the elements. This method for identifying the elements, to a large extent automates the whole process of creating an information system, starting from the business requirements. Business analysis and software development come closer to each other than before. With this method, not only an agile information system is developed but it can also be developed *fast*, reducing time for changes as well as initial time-to-market.

As Martin Op't Land pointed out in his disseration, organization splits should be performed based on business functions [OtL08]. However, currently, organization splits are to a large extent slowed down or even determined by the information system(s) supporting the organization. With NS, an agility can be reached such that it should be able to cut off the part of the information system that is transfered to another or a new organization. However, with the method as described in this masters thesis, it is very clear exactly which part of the information system should be cut off: an organization boundary in the ontological model can be mapped onto a boundary in the information system. This agility in information systems does not only improve the agility of organizations in splitting, but also eases the allying with new partners [OtL08, p.102].

8.3 Future work

This thesis knows some limitations which are also issues for future work: First of all, it was not found how to model derived facts in a Normalized System. Several issues play a role here, among which are: whether the derived fact should be calculated on each use or stored for later re-use, and whether the value of the derived fact should change when an original fact it is based on changes.

In this research, three cross-cutting concerns were identified: persistency (for data), logging (for actions), and authorization (for both data and actions). Authorization should make clear only an authorized person can access data or perform some actions. Every organization has its own policy for authorization. However, it seems that authorization can be expressed rather formal: *who* may peform *which action* on *which instance* (data) at *what time*? It should be investigated how authorization can be formalized and can be modeled within a NS.

This research did not look into I- and D-transaction that should be available for exteral actors, e.g., track-and-trace information about a delivery. Since these tasks are identified as generic supporting tasks, they are no longer available as a single action and therefore not available for external actors. It may be worthwhile to look into cases whether these transactions are really necassary or whether the current interface already shows enough information to the external user.

In Section 7.4, it was shown how a second subsidy scheme could be added by using a new implementation, or version, of some actions and data elements. It should be investigated whether this is truly the best way or whether another approache is better. This could even

clear whether modelling different schemes in different DEMO models is a good approach or whether that should be done differently.

Furthermore, it is considered there should be a tool to support the generation of elements from a DEMO model as way for expanding the elements to real code (class files etc.); tool support is considered essential for the succes of an approach like this one. Tool support also makes it easier to make changes to an implemented system.

Also, it should be investigated whether an agile information system really contributes to enterprise agility. It is clear other factors also play a role in enterprise agility, such as people, but it not yet clear which factors really define an enterprise's agility, Moreover, it should be investigated how the NS principles can be applied to these factors.

Appendices

A. Legends of the DEMO aspect models

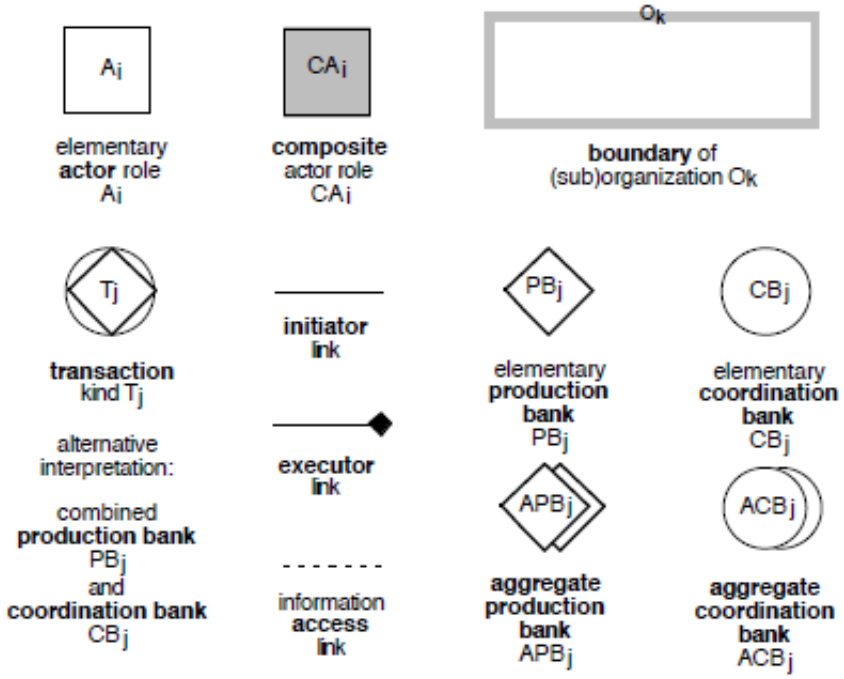


Figure A.1: Legend of the OCD

Listing A.1: Basic action rule syntax

```
1  when < transaction kind > of < object identifier(s) > is < transaction status > *
2      if < boolean logical expression>
3      then < transaction kind > of < object identifier(s) > must be < perfective form of C–act> **
4      else < transaction kind > of < object identifier(s) > must be < perfective form of C–act> **

   * < transaction status (C–fact) > = requested, promised, executed, stated, accepted, etc.
   ** < perfective form of C–act > = requested, promised, executed, stated, accepted, etc.
```

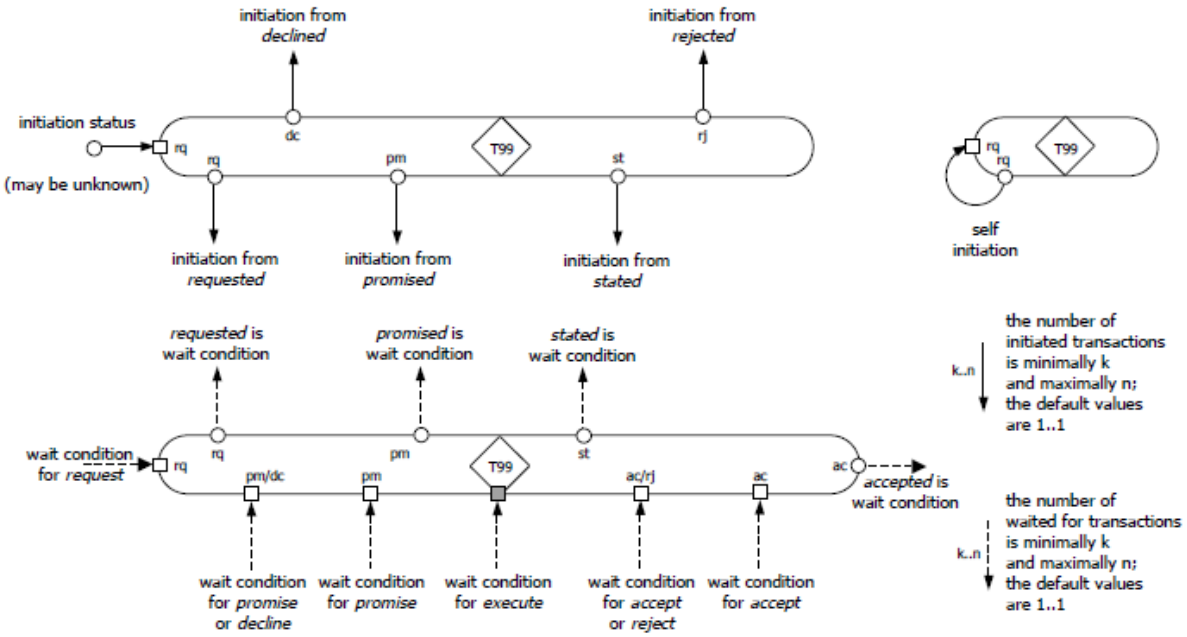


Figure A.2: Legend of the PSD

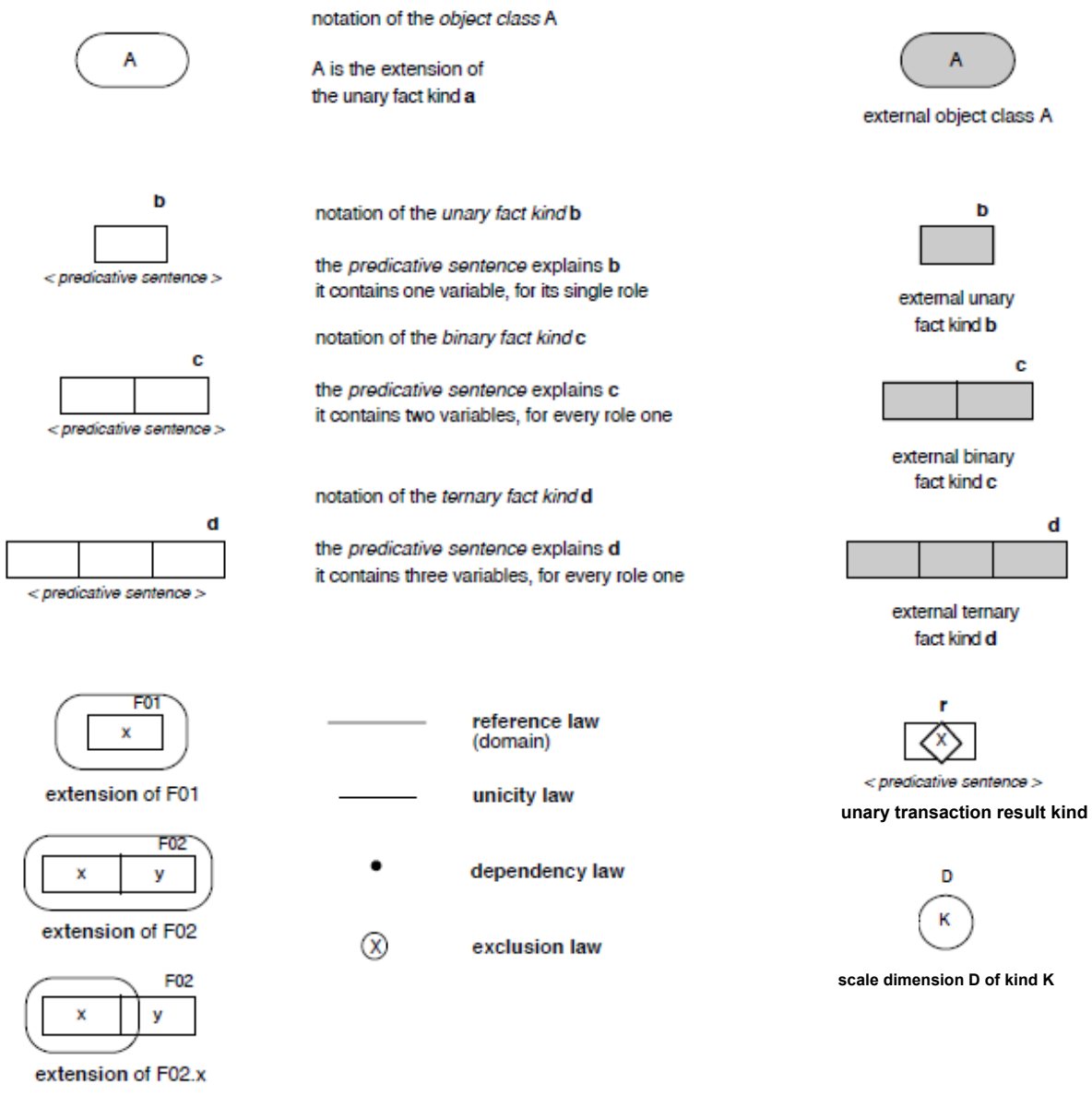


Figure A.3: Legend of the SSD

B. Workflows

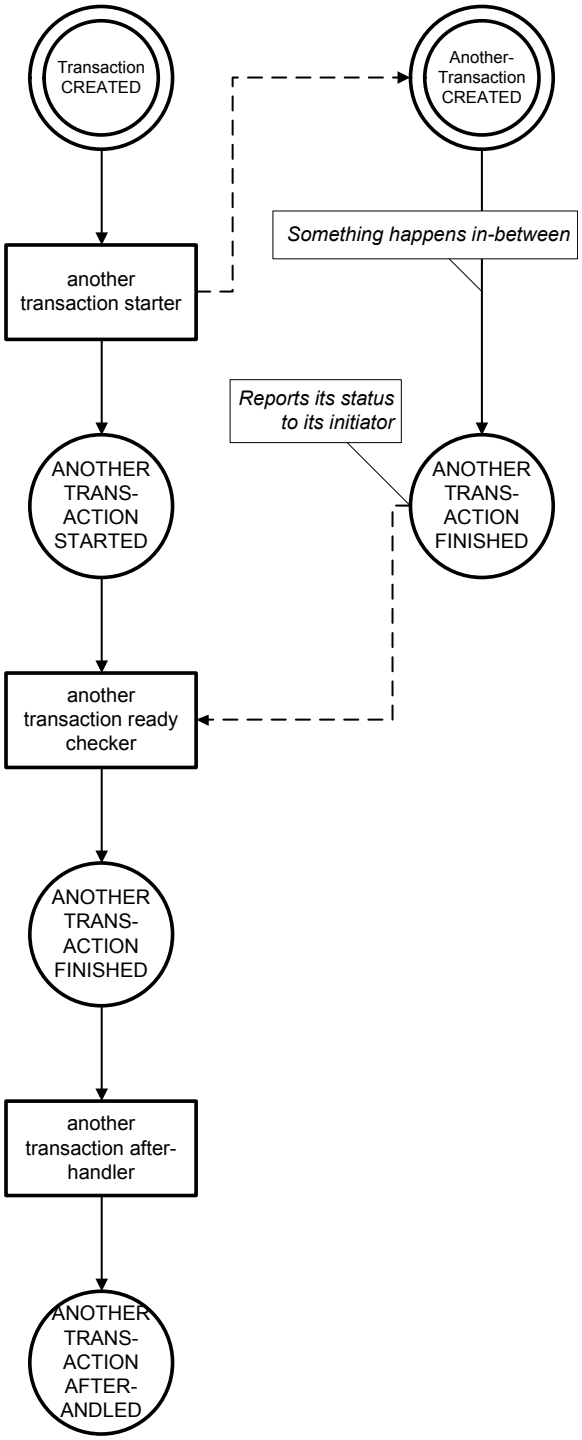


Figure B.1: Workflows for the nesting of a single instantiation of another transaction

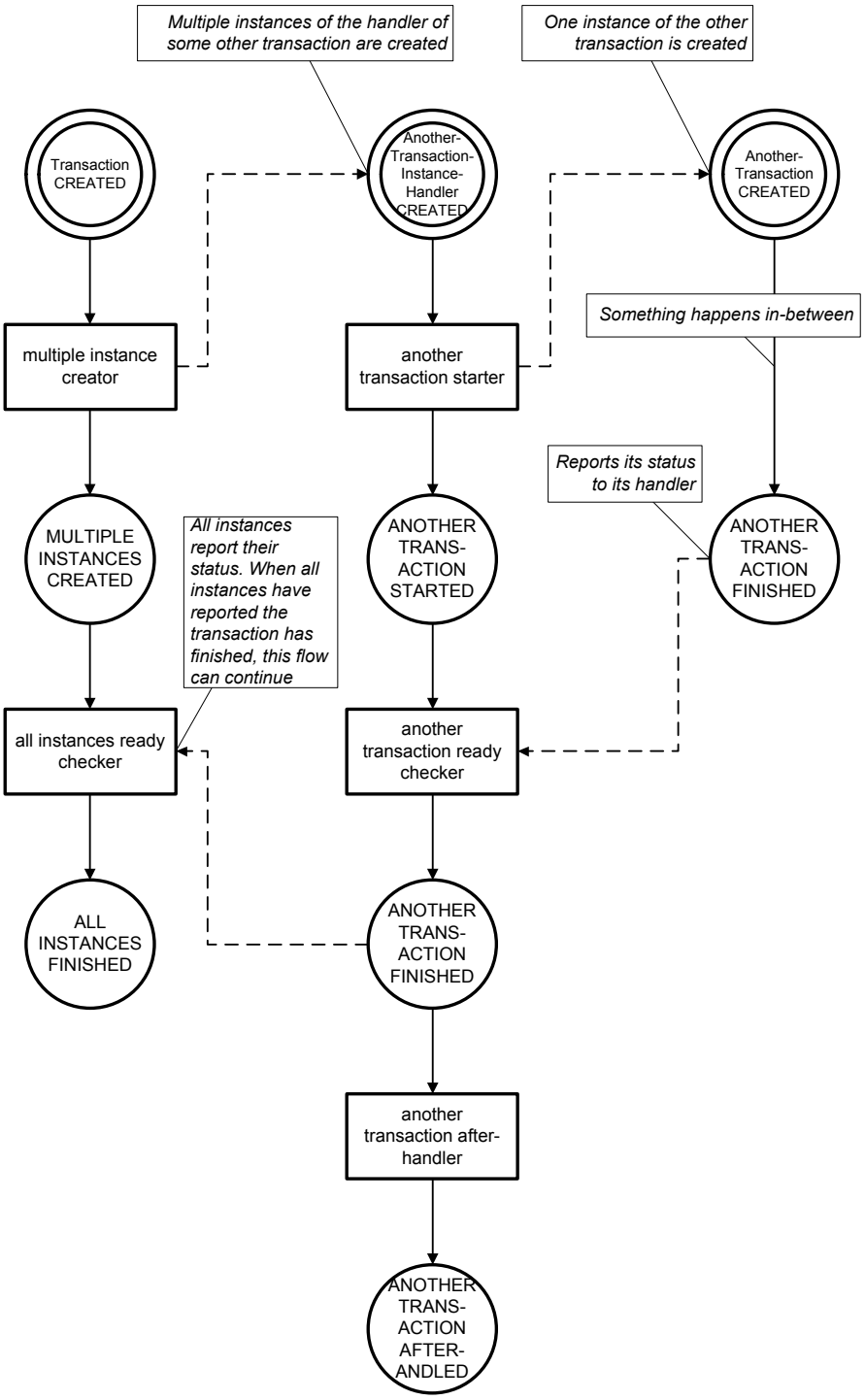


Figure B.2: Workflows for the nesting of multiple instantiations of another transaction

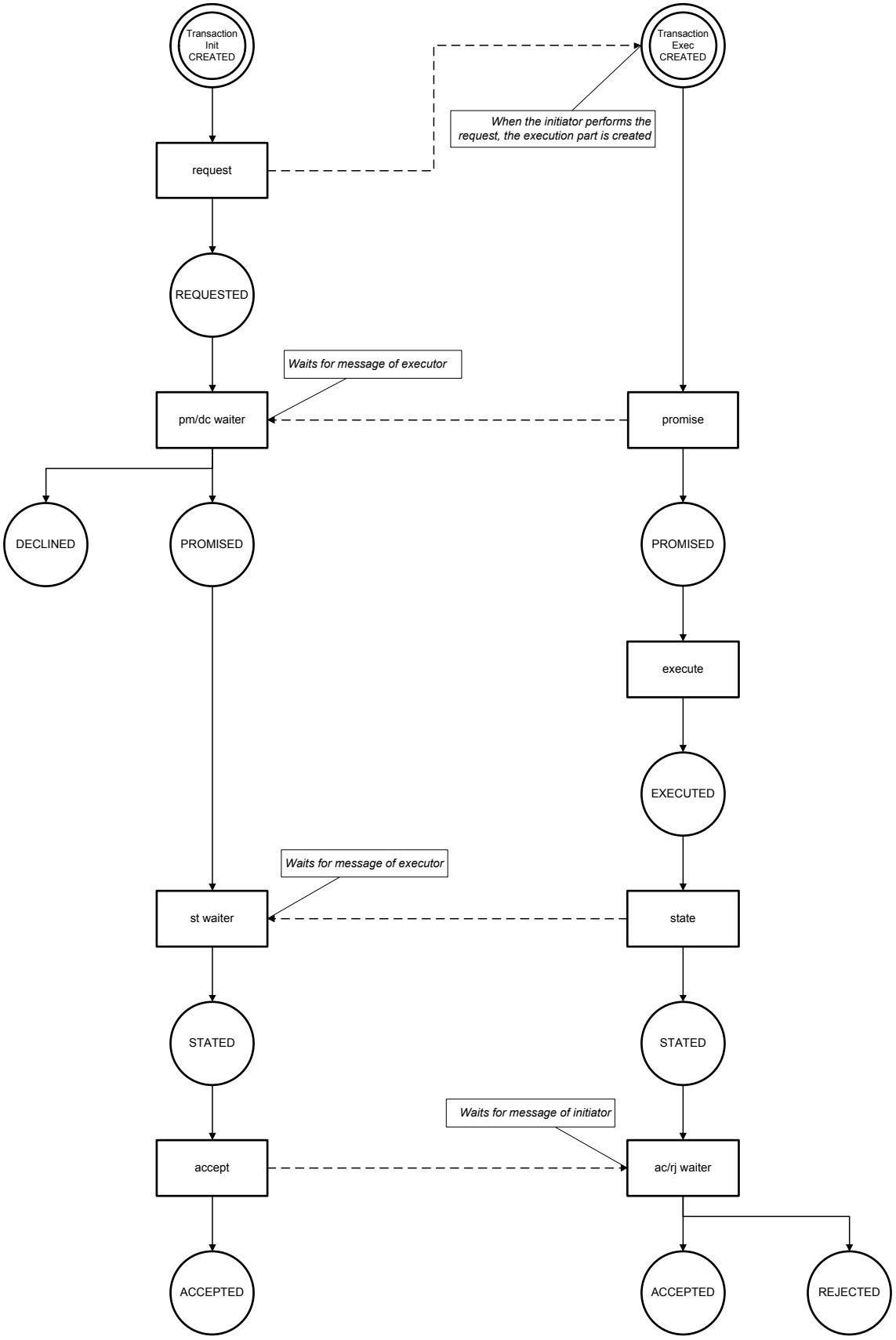


Figure B.3: Basic workflows for the two transaction elements and the communication between them

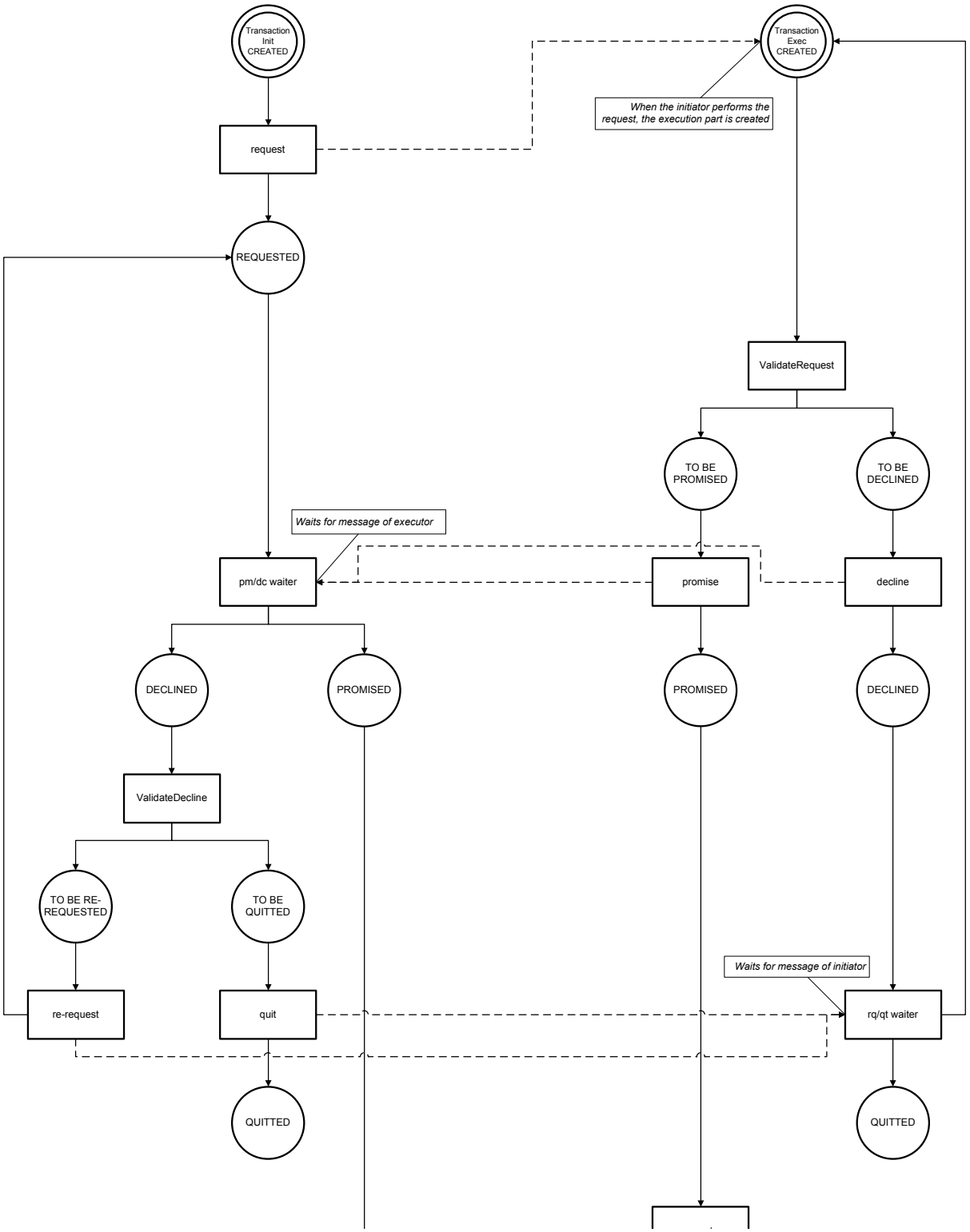


Figure B.4: Complete workflows for the two transaction elements and the communication between them

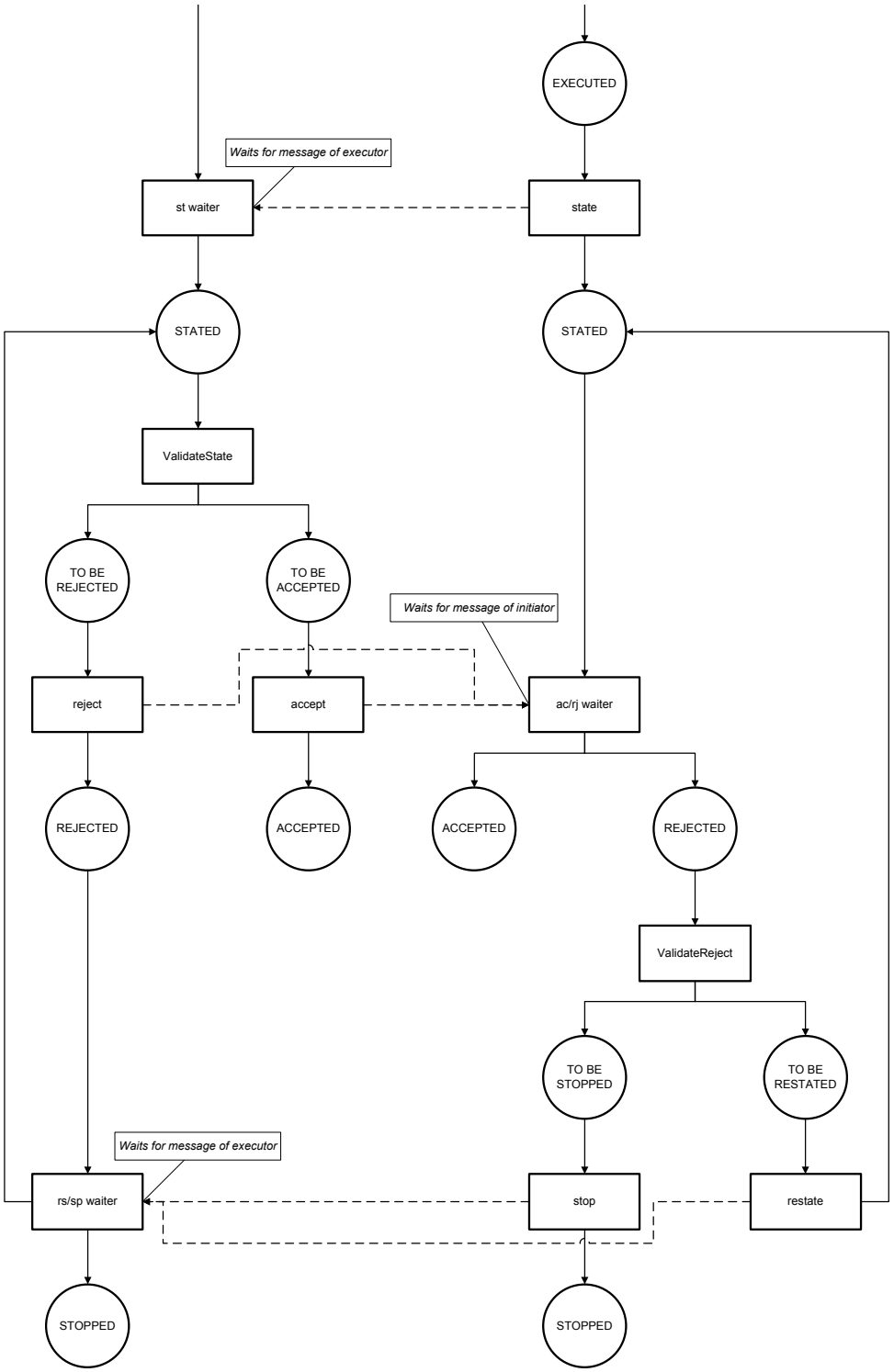


Figure B.4: Complete workflows for the two transaction elements and the communication between them (ctd.)

C. PIZZERIA DEMO MODELS

These models are taken from Dietz [Die06] for illustration. For the complete set of models as well as a complete description of the pizza case, the reader is referred to that same book.

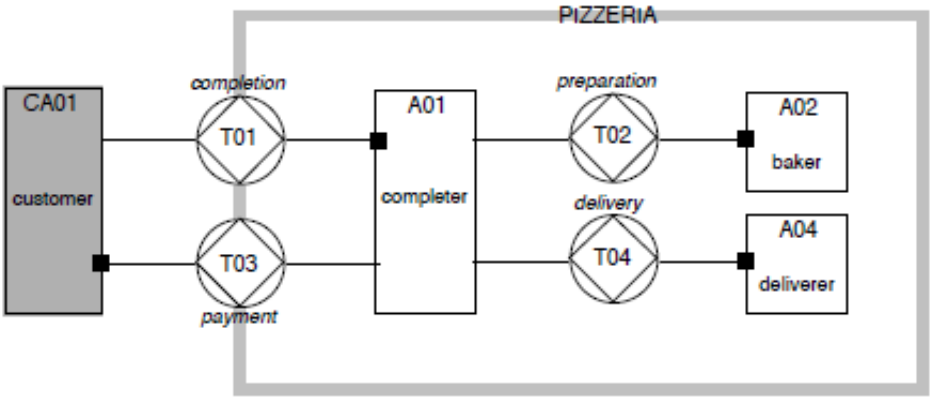


Figure C.1: Detailed ATD of the pizzeria

transaction kind	**result kind**
T01 completion	R01 [purchase] has been completed
T02 preparation	R02 [purchase] has been prepared
T03 payment	R03 [purchase] has been paid
T04 delivery	R04 [purchase] has been delivered

Table C.1: TRT of the pizzeria

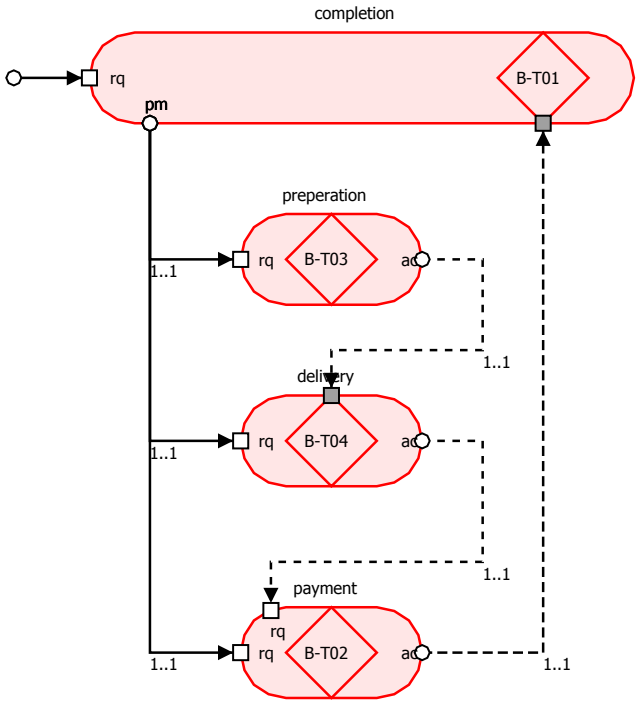


Figure C.2: PSD of the pizzeria

D. Government Subsidy Schemes: Sloopregeling - DEMO models

The DEMO models were created by Cor Nagtegaal (cor.nagtegaal@capgemini.com), Martin Op't Land (martin.optland@capgemini.com), Jan van Santbrink (jan.van.santbrink@capgemini.com), and Marien Krouwel (marien.krouwel@capgemini.com).

Both object classes vehicle and subsidy knows derived facts. A subsidy has rules for calculating the subsidy amount and whether the application satisfies the conditions for granting. For the subsidy object class, the derivation rules are stated in the FDL (Listing D.9). For the vehicle object class, which is an external object class, the properties are listed in Table D.2. These are not depicted in the FDL because that would make it too voluminous.

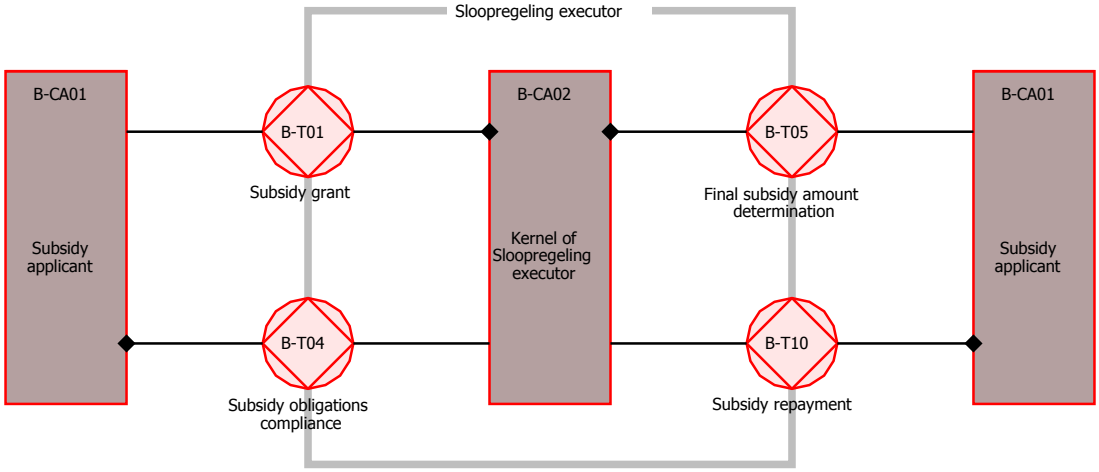


Figure D.1: Kernel of the sloopregeling

transaction kind	**result kind**
T01 subsidy grant	R01 [subsidy] has been granted
T02 formal evaluation	R02 [subsidy] has been formally evaluated
T03 substantive evaluation	R03 [subsidy] has been substantively evaluated
T04 subsidy obligations compliance	R04 obligations for [subsidy] have been met
T05 final subsidy amount determination	R05 final amount for [subsidy] has been determined
T06 subsidy payment	R06 [subsidy] has been paid out
T07 random check	R07 random check for [period] has been performed
T08 inspection	R08 [subsidy] has been inspected
T09 subsidy recovery	R09 [subsidy] has been recovered
T10 subsidy repayment	R10 [subsidy] has been repaid

Table D.1: TRT of the sloopregeling

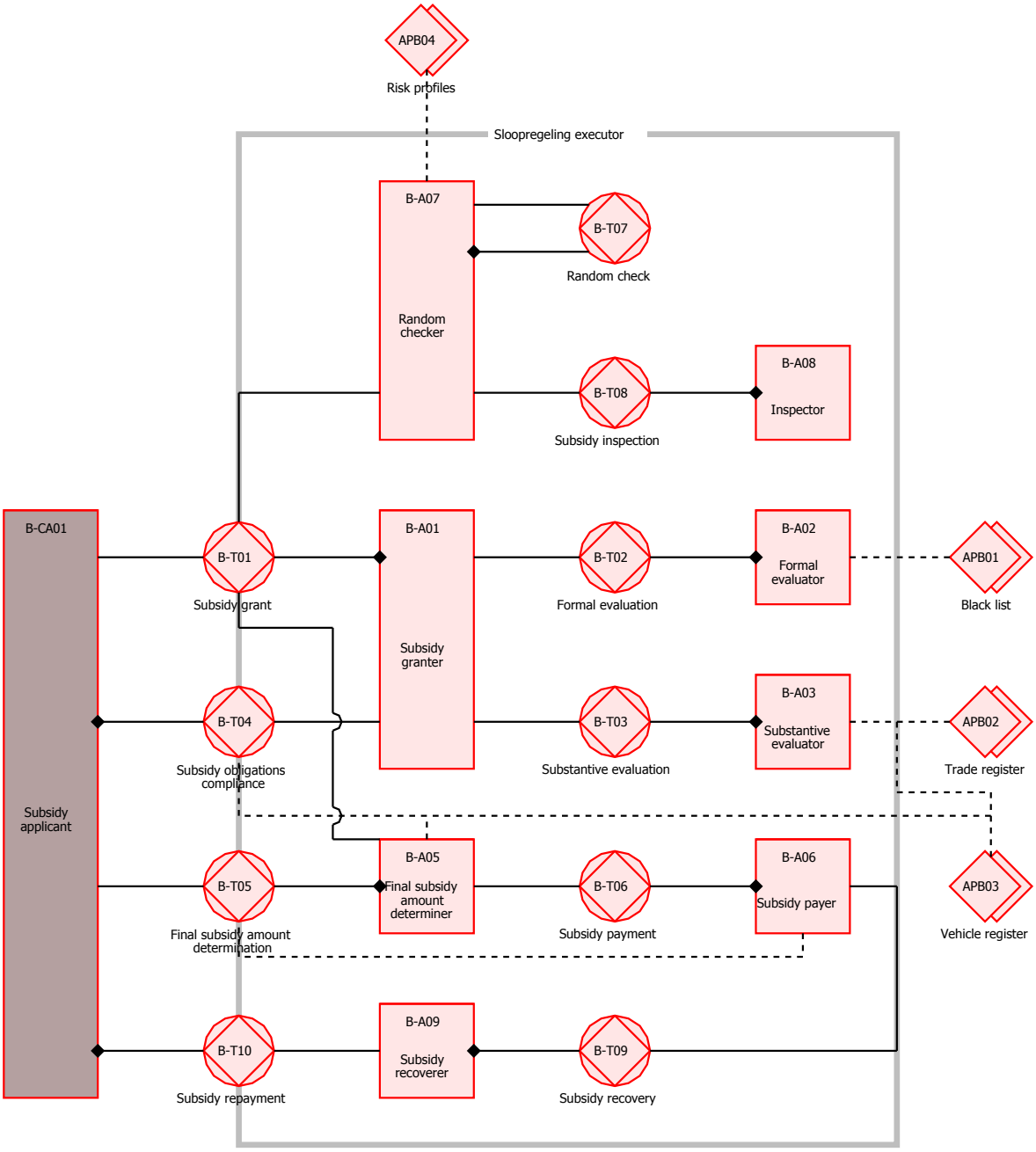
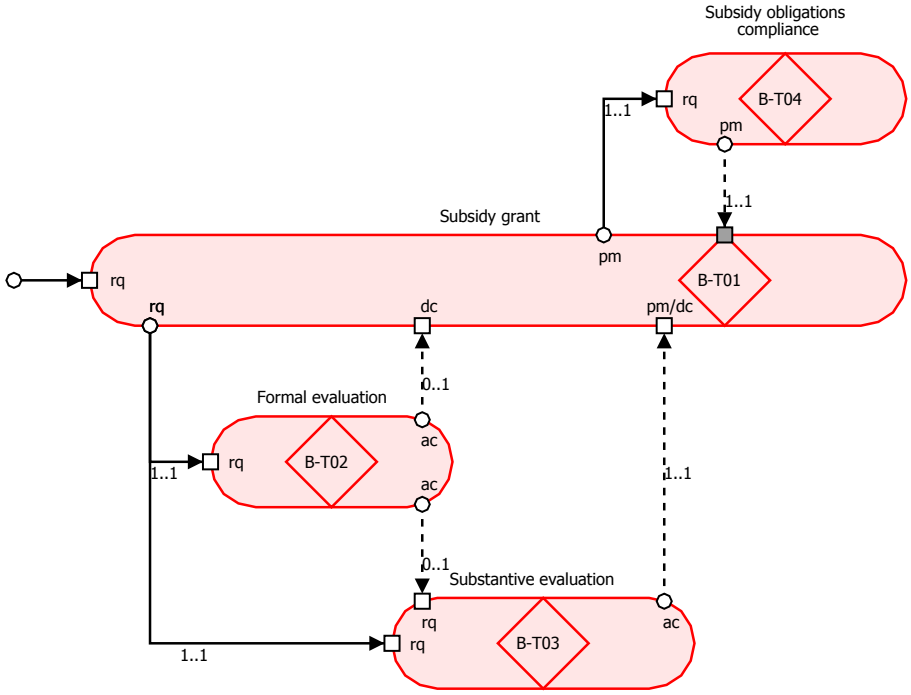
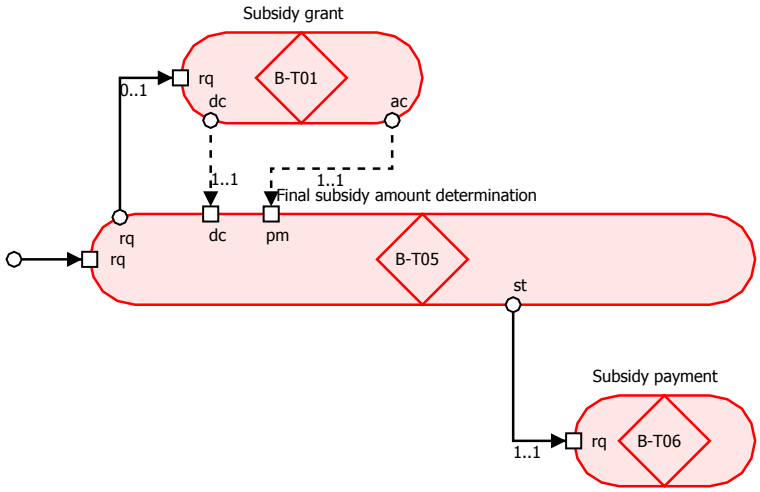


Figure D.2: OCD of the sloopregeling

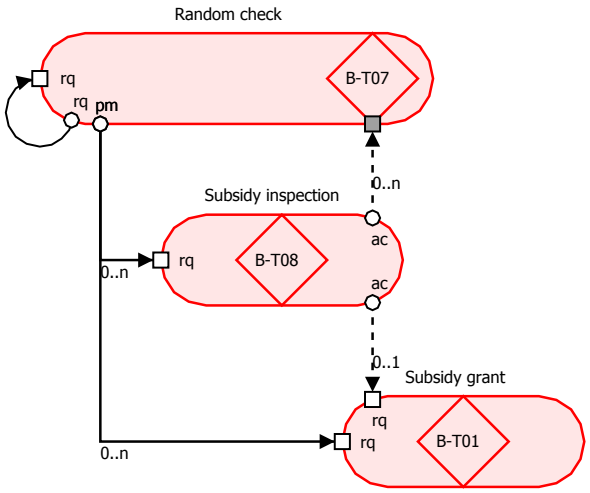


(a) PSD sloopregeling subsidy grant

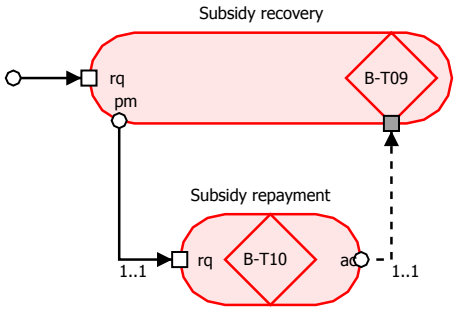


(b) PSD sloopregeling subsidy amount determination

Figure D.3: PSDs of the sloopregeling



(c) PSD sloopregeling inspection



(d) PSD sloopregeling subsidy recovery

Figure D.3: PSDs of the sloopregeling (ctd.)

Listing D.1: Action rules for actor role A01 Subsidy granter

```
1   when T01 subsidy grant of [subsidy] is requested
2        T02 formal evaluation of [subsidy] must be requested

4   when T02 formal evaluation of [subsidy] is stated
5        T02 formal evaluation of [subsidy] must be accepted

7   when T02 formal evaluation of [subsidy] is accepted
8        if formal evaluation result of [subsidy] = ok
9        then T03 substantive evaluation of [subsidy] must be requested
10       else T01 subsidy grant of [subsidy] must be declined

12  when T03 substantive evaluation of [subsidy] is stated
13       T03 substantive evaluation of [subsidy] must be accepted

15  when T03 substantive evaluation of [subsidy] is accepted
16       if substantive evaluation result of [subsidy] = ok
17       then T01 subsidy grant of [subsidy] must be promised
18       else T01 subsidy grant of [subsidy] must be declined

20  when T01 subsidy grant of [subsidy] is promised
21       T04 subsidy obligations compliance of [subsidy] must be requested

23  when T04 subsidy obligations compliance of [subsidy] is promised
24       T01 subsidy grant of [subsidy] must be executed

26  when T01 subsidy grant of [subsidy] is executed
27       T01 subsidy grant of [subsidy] must be stated

29  when T04 subsidy obligations compliciance of [subsidy] is stated
30       T04 subsidy obligations compliciance of [subsidy] must be accepted
```

Listing D.2: Action rules for actor role A02 Formal evaluator

```
1  when T02 formal evaluation of [subsidy] is requested
2       T02 formal evaluation of [subsidy] must be promised

4  when T02 formal evaluation of [subsidy] is promised
5       T02 formal evaluation of [subsidy] must be executed

7  when T02 formal evaluation of [subsidy] is executed
8       T02 formal evaluation of [subsidy] must be stated
```

Listing D.3: Action rules for actor role A03 Substantive evaluator

```
1  when T03 substantive evaluation of [subsidy] is requested
2       T03 substantive evaluation of [subsidy] must be promised

4  when T03 substantive evaluation of [subsidy] is promised
5       T03 substantive evaluation of [subsidy] must be executed

7  when T03 substantive evaluation of [subsidy] is executed
8       T03 substantive evaluation of [subsidy] must be stated
```

Listing D.4: Action rules for actor role A05 Final subsidy amount determiner

```
1  when T05 final subsidy amount determination of [subsidy] is requested
2       if subsidy grant has been requested
3       then if [subsidy] has been granted
4            then T05 final subsidy amount determination must be promised
5            else if subsidy grant has been declined
6                 then T05 final subsidy amount determination must be declined
7       else T01 subsidy grant of [subsidy] must be requested

9  when T01 subsidy grant of [subsidy] is stated
10      T01 subsidy grant of [subsidy] must be accepted

12 when T01 subsidy grant of [subsidy] is accepted
13      T05 final subsidy amount determination must be promised

15 when T01 subsidy grant of [subsidy] is declined
16      T05 final subsidy amount determination must be declined

18 when T05 final subsidy amount determination of [subsidy] is promised
19      T05 final subsidy amount determination of [subsidy] must be executed

21 when T05 final subsidy amount determination of [subsidy] is executed
22      T05 final subsidy amount determination of [subsidy] must be stated

24 when T05 final subsidy amount determination of [subsidy] is stated
25      T06 subsidy payment of [subsidy] must be requested

27 when T06 subsidy payment of subsidy] is stated
28      T06 subsidy payment of [subsidy] must be accepted
```

Listing D.5: Action rules for actor role A06 Subsidy payer

```
1   when T06 subsidy payment of [subsidy] is requested
2        T06 subsidy payment of [subsidy] must be promised

4   when T06 subsidy payment [subsidy] is promised
5        T06 subsidy payment of [subsidy] must be executed

7   when T06 subsidy payment of [subsidy] is executed
8        T06 subsidy payment of [subsidy] must be stated

10  when T06 subsidy payment of [subsidy] is cancelled
11       T09 subsidy recovery of [subsidy] must be requested

13  when T09 subsidy recovery of [subsidy] is stated
14       T09 subsidy recovery if [subsidy] must be accepted
```

Listing D.6: Action rules for actor role A07 Random checker

```
1   when T07 random check of [period] is requested
2        T07 random check of [period] must be promised

4   when T07 random check of [period] is promised
5        for each [subsidy] that matches current risk profile
6                 and hasn't been checked yet
7            T08 subsidy inspection of [subsidy] must be requested

9   when T08 subsidy inspection of [subsidy] is stated
10       if inspection was performed properly
11       then T08 subsidy inspection must be accepted
12       else T08 subsidy inspection must be rejected

14  when T08 subsidy inspection of [subsidy] is accepted
15       if inspection result subsidy does not equal original subsidy
16       then T01 of [subsidy] must be cancelled
17            T01 of [inspection result subsidy] must be requested

19  when T01 subsidy grant of [subsidy] is stated
20       T01 subsidy grant of [subsidy] must be accepted

22  when all T08's in random check of [period] have been accepted
23       T07 random check of [period] must be executed

25  when T07 random check of [period] is executed
26       T07 random check of [period] must be stated

28  when T07 random check of [period] is stated
29       T07 random check of [period] must be accepted
```

Listing D.7: Action rules for actor role A08 Inspector

```
1   when T08 inspection of [subsidy] is requested
2        T08 inspection of [subsidy] must be promised

4   when T08 inspection [subsidy] is promised
5        T08 inspection of [subsidy] must be executed

7   when T08 inspection of [subsidy] is executed
8        T08 inspection of [subsidy] must be stated
```

Listing D.8: Action rules for actor role A09 Subsidy recoverer

```
1   when T09 subsidy recovery of [subsidy] is requested
2        T09 subsidy recovery of [subsidy] must be promised

4   when T09 subsidy recovery [subsidy] is promised
5        T10 subsidy repayment of [subsidy] must be requested

7   when T10 subsidy repayment [subsidy] is stated
8        if repaid amount of [subsidy] = final subsidy amount of [subsidy]
9        then T10 subsidy repayment of [subsidy] must be accepted
10       else T10 subsidy repayment of [subsidy] must be rejected

12  when T10 subsidy repayment [subsidy] is accepted
13       T09 subsidy recovery of [subsidy] must be executed

15  when T09 subsidy recovery of [subsidy] is executed
16       T09 subsidy recovery of [subsidy] must be stated
```

property	**scale dimension**	**kind**	**internal type**	**range**
first registration date	time	interval	date	
empty weight	weight	ratio	integer	
car type	car type	nominal	string	person, company
fuel type	fuel type	nominal	string	gasoline, Liquid Petrol Gas (LPG), diesel, Compressed Natural Gas (CNG), hydrogen, electric
last ascription date	time	interval	date	
inspection expriy date	time	interval	date	
status	status	nominal	string	valid, invalid, disassembled, wait for inspection, exported, unknown
LPG/g3 (clean emission installation)	??	nominal	boolean	yes, no
black smoke filter	??	nominal	boolean	yes, no
emission limit compliance	??	nominal	boolean	yes, no

Table D.2: Properties of a vehicle (data can be acquired at the RDW)

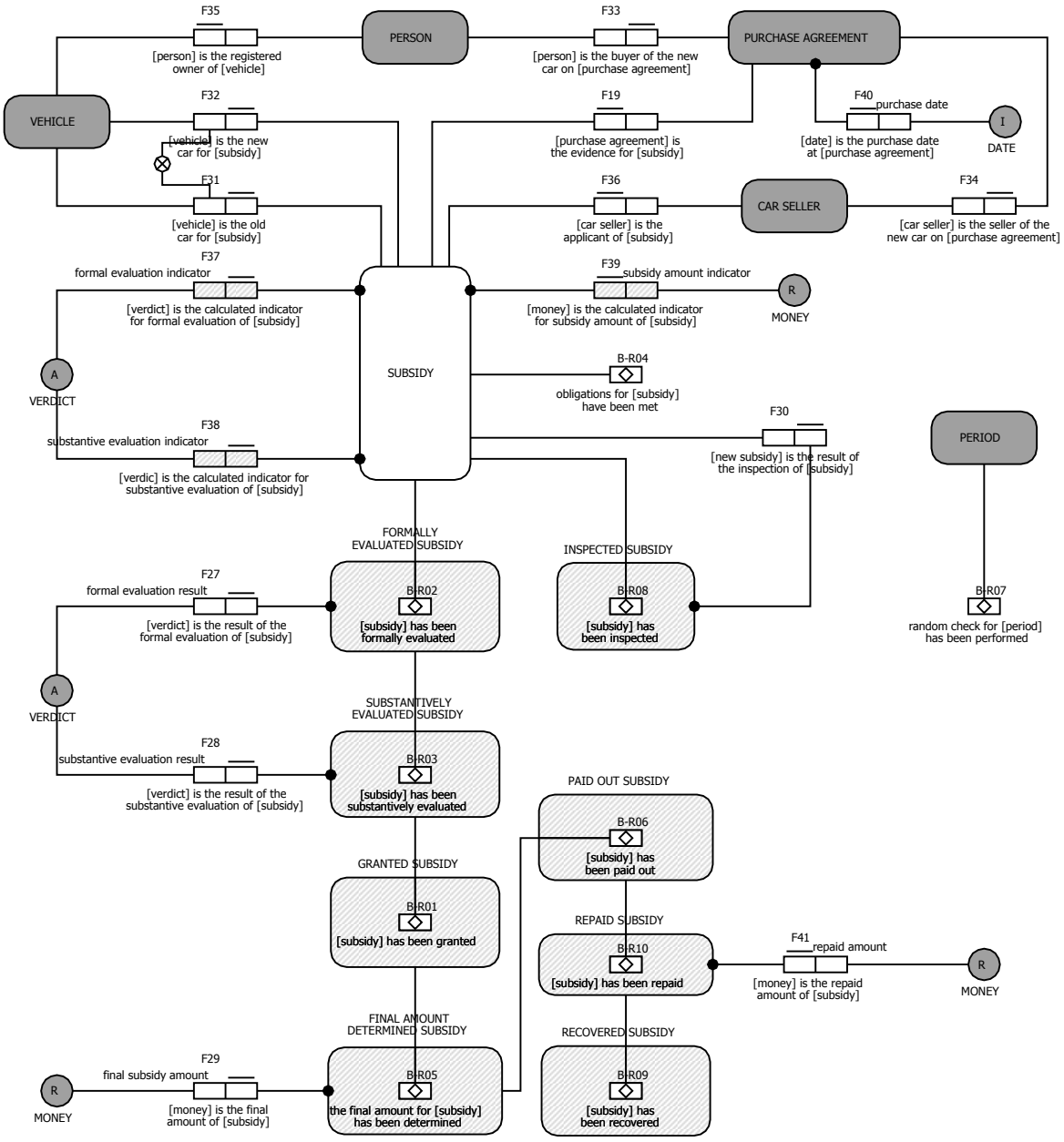


Figure D.4: SSD of the sloopregeling

Listing D.9: FDL of the sloopregeling

```
subsidy amount of [subsidy] =
    if fuel_type of [old car] of [subsidy] = (gasoline or lpg)
            and first_registration_date of [old car] of [subsidy] < 01-01-1990
    then 750
    else if fuel_type of [old car] of [subsidy] = (gasoline or lpg)
            and first_registration_date of [old car] of [subsidy] >= 01-01-1990
            and first_registration_date of [old car] of [subsidy] < 01-01-1996
    then 1000
    else if type of [old car] of [subsidy] = person
            and fuel_type of [old car] of [subsidy] = diesel
            and first_registration_date of [old car] of [subsidy] < 01-01-2000
    then 1000
    else if type of [old car] of [subsidy] = company
            and fuel_type of [old car] of [subsidy] = diesel
            and first_registration_date of [old car] of [subsidy] < 01-01-2000
            and empty_weight < 1800
    then 1000
    else if type of [old car] of [subsidy] = company
            and fuel_type of [old car] of [subsidy] = diesel
            and first_registration_date of [old car] of [subsidy] < 01-01-2000
            and empty_weight >= 1800
    then 1750

formal evaluation indicator of [subsidy] =
    not subsidy cealing is reached
    and not last_ascription_date of [old car] of [subsidy] > 01-03-2008
    and not inspection_expiry_date of [old car] of [subsidy] -
            purchase_date of [purchase agreement] of [subsidy] < 3 months
    and not status of [old car] of [subsidy] = (invalid or wait_for_inspection)
    and not subsidy for old car already paid out
    and purchase_date of [purchase agreement] of [subsidy] >= 29-05-2009

informal evaluation indicator of [subsidy] =
  ( type of [old car] of [subdidy] = (person or company)
    and fuel_type of [old car] of [subsidy] = (gasoline or lpg)
    and first_registration_date of [old car] of [subsidy] < 01-01-1996
    or type of [old car] of [subdidy] = (person or company)
    and fuel_type of [old car] of [subsidy] = diesel
    and first_registration_date of [old car] of [subsidy] < 01-01-2000
  ) and
  ( type of [new car] of [subdidy] = (person or company)
    and fuel_type of [new car] of [subsidy] = gasoline
    and first_registration_date of [new car] of [subsidy] > 31-12-2000
    or type of [new car] of [subdidy] = (person or company)
    and fuel_type of [new car] of [subsidy] = lpg
    and lpgg3 of [new car] of [subsidy] = yes
    and first_registration_date of [new car] of [subsidy] < 31-12-2000
    or type of [new car] of [subdidy] = (person or company)
    and fuel_type of [new car] of [subsidy] = diesel
    and black_smoke_filter of [new car] of [subsidy] = yes
    or type of [new car] of [subdidy] = (person or company)
    and fuel_type of [new car] of [subsidy] = (electricity or CNG or hydrogen)
    or type of [new car] of [subdidy] = company
    and emission_limit_compliance of [new car] of [subsidy] = yes
  )
```

E. Government Subsidy Schemes: Sloopregeling - NS elements

Data elements

MailSender
`String message`

Payer
`Integer amount`

Subsidy
`CarSeller applicant`
`PurchaseAgreement purchaseAgreement`
`Car oldCar`
`Car newCar`

Period
`Integer month`
`Integer year`

Vehicle
`String numberplate`

Person
`Integer bsn`

PurchaseAgreement
`Date purchaseDate`
`Car newCar`
`Car oldCar`
`CarSeller carSeller`
`Person carBuyer`

CarSeller
`Integer tradeRegistrationNumber`

Verdict
`Boolean verdict`

Money
`Float amount`

FormallyEvaluatedSubsidy
`Subsidy subsidy`

SubstantivelyEvaluatedSubsidy
`FormallyEvaluatedSubsidy subsidy`

GrantedSubsidy
`SubstantivelyEvaluatedSubsidy subsidy`

FinalAmountDeterminedSubsidy
`GrantedSubsidy subsidy`

PaidOutSubsidy
`FinalAmountDeterminedSubsidy subsidy`

RepaidSubsidy
`PaidOutSubsidy subsidy`

RecoveredSubsiy
`RepaidSubsidy subsidy`

InspectedSubsidy
`Subsidy subsidy`

SubsidyGrantInit
`Subsidy subsidy`

SubsidyGrantExec
`Subsidy subsidy`

SubsidyFormalEvaluationInit
`Subsidy subsidy`

SubsidyFormalEvaluationExec
`Subsidy subsidy`

SubsidySubstantiveEvaluationInit
`Subsidy subsidy`

SubsidySubstantiveEvaluationExec
`Subsidy subsidy`

SubsidyObligationsComplianceInit
`Subsidy subsidy`

SubsidyFinalAmountDeterminationExec
`Subsidy subsidy`

SubsidyPaymentInit
`Subsidy subsidy`

SubsidyPaymentExec
`Subsidy subsidy`

RandomCheckInit
`Period period`

RandomCheckExec
`Period period`

RandomCheckExecInspectionHandler
`Subsidy subsidy`

SubsidyInspectionInit
`Subsidy subsidy`

SubsidyInspectionExec
`Subsidy subsidy`

SubsidyRecoveryInit
`Subsidy subsidy`

SubsidyRecoveryExec
`Subsidy subsidy`

SubsidyRepaymentInit
`Subsidy subsidy`

Action elements

MailSenderCompose
`Standard: composes a message based on some standard messages and the creator of the data element`

MailSenderPrint
`Bridge Printer (not elaborated on)`

MailSenderSend
`Manual`

PayerPay
`Standard: pay using some bank application`

SubsidyOpWaiter
`External`

SubsidyPmWaiter
`External`

SubsidyGrantInitRequest
`Bridge SubsidyGrantExec`

SubsidyGrantInitStWaiter
`External`

SubsidyGrantInitPmDcWaiter
`External`

SubsidyGrantInitAccept
`Standard: send message to executor (flow)`

SubsidyGrantExecT02Creator
`Bridge SubsidyFormalEvaluationInit`

SubsidyGrantExecT02AcWaiter
`External`

SubsidyGrantExecCheckFormalEvaluationResult
`Standard: Listing E.1`

SubsidyGrantExecT03Creator
`Bridge SubsidySubstantiveEvaluationInit`

SubsidyGrantExecT03AcWaiter

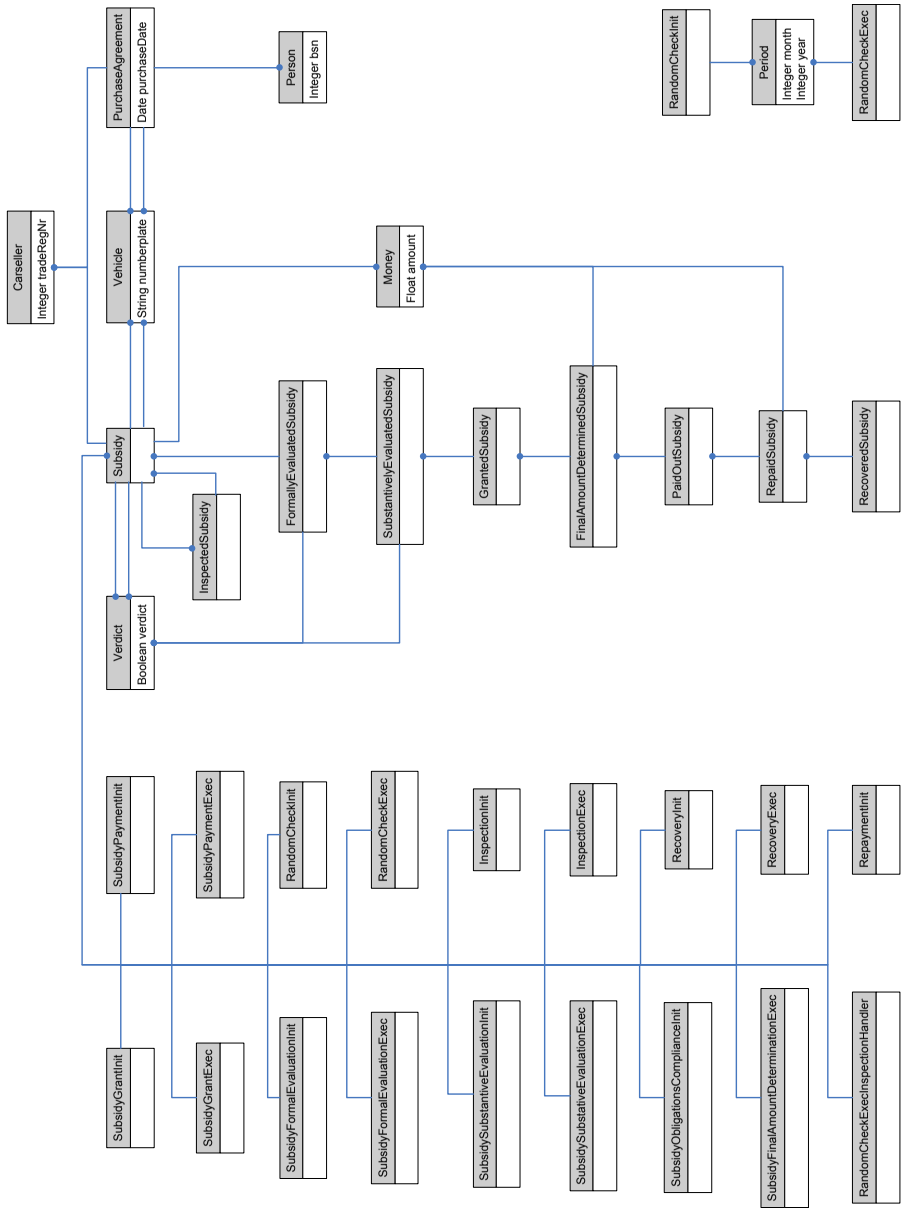


Figure E.1: ER diagram for the data elements

External

SubsidyGrantExecCheckSubstantiveEvaluationResult
Standard: Listing E.2

SubsidyGrantExecPromise
Standard: do nothing

SubsidyGrantExecDecline
Bridge MailSender

SubsidyGrantExecRqQtWaiter
External

SubsidyGrantExecT04Creator
Bridge SubsidyObligationsComplianceInit

SubsidyGrantExecT04PmWaiter
External

SubsidyGrantExecExecute
Standard: do nothing

SubsidyGrantExecState
Bridge MailSender

SubsidyGrantExecAcRjWaiter
External

SubsidyFormalEvaluationInitRequest
Bridge SubsidyFormalEvaluationExec

SubsidyFormalEvaluationInitPmDcWaiter
External

SubsidyFormalEvaluationInitStWaiter
External

SubsidyFormalEvaluationInitAccept
Standard: send message to executor (flow)

SubsidyFormalEvaluationExecPromise
Standard: send message to initiator (flow)

Listing E.1: Code for SubsidyGrantExecCheckFormalEvaluationResult

```
if FormalEvaluationResult of [subsidy] = ok
   then FormalEvaluationResultOk
   else FormalEvaluationResultNotOk
```

Listing E.2: Code for SubsidyGrantExecCheckSubstantiveEvaluationResult

```
if SubstantiveEvaluationResult of [subsidy] = ok
   then SubstantiveEvaluationResultOk
   else SubstantiveEvaluationResultNotOk
```

SubsidyFormalEvaluationExecExecute
`Standard: Listing E.3`

SubsidyFormalEvaluationExecState
`Standard: send message to initiator (flow)`

SubsidyFormalEvaluationAcRjWaiter
`External`

SubsidySubstantiveEvaluationInitRequest
`Bridge SubsidySubstantiveEvaluationExec`

SubsidySubstantiveEvaluationInitPmDcWaiter
`External`

SubsidySubstantiveEvaluationInitStWaiter
`External`

SubsidySubstantiveEvaluationInitAccept
`Standard: send message to executor (flow)`

SubsidySubstantiveEvaluationExecPromise
`Standard: send message to initiator (flow)`

SubsidySubstantiveEvaluationExecExecute
`Standard: Listing E.4`

SubsidySubstantiveEvaluationExecState
`Standard: send message to initiator (flow)`

SubsidySubstantiveEvaluationAcRjWaiter
`External`

SubsidyObligationsComplianceInitRequest
`Standard: do nothing`

SubsidyObligationsComplianceInitPmDcWaiter
`Standard: do nothing`

SubsidyObligationsComplianceInitStWaiter
`External`

SubsidyObligationsComplianceInitAccept
`Standard: do nothing`

SubsidyFinalAmountDeterminationExecCheckGrantHasBeenRequested

Listing E.3: Code for SubsidyFormalEvaluationExecExecute

```
FormalEvaluationResult of [subsidy] := FormalEvaluationIndicator of [subsidy]
```

Listing E.4: Code for SubsidySubstantiveEvaluationExecExecute

```
SubstantiveEvaluationResult of [subsidy] := SubstantiveEvaluationIndicator of [subsidy]
```

Standard: Listing E.6

SubsidyFinalAmountDeterminationExecCheckSubsidyHasBeenGranted
Standard: Listing E.7

SubsidyFinalAmountDeterminationExecCheckGrantHasBeenDeclined
Standard: Listing E.8

SubsidyFinalAmountDeterminationExecT01Creator
Bridge SubsidyGrantInit

SubsidyFinalAmountDeterminationExecT01PmAcWaiter
External

SubsidyFinalAmountDeterminationExecT04State
Standard: send message to T04 (flow)

SubsidyFinalAmountDeterminationExecPromise
Standard: do nothing

SubsidyFinalAmountDeterminationExecDecline
Bridge MailSender

SubsidyFinalAmountDeterminationExecRqQtWaiter
External

SubsidyFinalAmountDeterminationExecExecute
Standard: Listing E.5

SubsidyFinalAmountDeterminationExecState
Bridge MailSender

SubsidyFinalAmountDeterminationExecT06Creator
Bridge SubsidyPaymentInit

SubsidyFinalAmountDeterminationExecAcRjWaiter
External

SubsidyPaymentInitRequest

Listing E.5: Code for SubsidyFinalAmountDeterminationExecExecute

```
FinalSubsidyAmount of [subsidy] := SubsidyAmountIndicator of [subsidy]
```

Listing E.6: Code for SubsidyFinalAmountDeterminationExecCheckGrantHasBeenRequested

```
if exists T01 SubsidyGrant of [subsidy]
then GrantHasBeenRequested
else GrantHasNotBeenRequested
```

Listing E.7: Code for SubsidyFinalAmountDeterminationExecCheckSubsidyHasBeenGranted

```
if SubsidyGrant of [subsidy] has been accepted
then SubsidyHasBeenGranted
else SubsidyHasNotBeenGranted
```

Bridge SubsidyPaymentExec

SubsidyPaymentInitPmDcWaiter
External

SubsidyPaymentInitStWaiter
External

SubsidyPaymentInitAccept
Standard: send message to executor (flow)

SubsidyPaymentExecPromise
Standard: send message to initiator (flow)

SubsidyPaymentExecExecute
Bridge Payer

SubsidyPaymentExecState
Standard: send message to initiator (flow)

SubsidyPaymentExecAcRjWaiter
External

SubsidyPaymentExecT09Creator
Bridge SubsidyRecoveryInit

RandomCheckInitRequest
Bridge RandomCheckExec

RandomCheckInitPmDcWaiter
External

RandomCheckInitStWaiter
External

RandomCheckInitAccept
Standard: send message to executor (flow)

RandomCheckExecPromise
Standard: send message to initiator (flow)

RandomCheckExecInspectionHandlersCreator
Standard: creates for each subsidy that matches the current risk profile and hasn't been checked yet, an InspectionHandler

RandomCheckExecInspectionAllAcWaiter
External: waits until all created Inspection handlers have reported the inspection transaction has been accepted

Listing E.8: Code for SubsidyFinalAmountDeterminationExecCheckGrantHasBeenDeclined

```
if SubidyGrant of [subsidy] has been declined
then GrantHasBeenDeclined
else GrantHasNotBeenDeclined
```

RandomCheckExecExecute
`Standard: do nothing`

RandomCheckExecState
`Standard: send message to initiator (flow)`

RandomCheckExecAcRjWaiter
`External`

RandomCheckExecInspectionHandlerT08creator
`Bridge SubsidyInspectionInit`

RandomCheckExecInspectionHandlerT08AcWaiter
`External`

RandomCheckExecInspectionHandlerCheckInspectionResult
`Standard: Listing E.9`

RandomCheckExecInspectionHandlerT01Canceller
`Standard: cancels the flow of the T01 of this subsidy`

RandomCheckExecInspectionHandlerT01Creator
`Bridge SubsidyGrantInit`

SubsidyInspectionInitRequest
`Bridge SubsidyInspectionExec`

SubsidyInspectionInitPmDcWaiter
`External`

SubsidyInspectionInitStWaiter
`External`

SubsidyInspectionInitCheckInspection
`Manual`

SubsidyInspectionInitAccept
`Standard: send message to executor (flow)`

SubsidyInspectionInitReject
`Standard: send message to executor (flow)`

SubsidyInspectionInitRsSpWaiter
`External`

SubsidyInspectionExecPromise
`Standard: send message to initiator (flow)`

SubsidyInspectionExecExecute

Listing E.9: Code for RandomCheckExecInspectionHandlerCheckInspectionResult

```
if [subsidy] = [inspection result subsidy] of [subsidy]
then InspectionResultOk
else InspectionResultNotOk
```

Manual

SubsidyInspectionExecState
Standard: send message to initiator (flow)

SubsidyInspectionExecAcRjWaiter
External

SubsidyRecoveryInitRequest
Bridge SubsidyRecoveryExec

SubsidyRecoveryInitPmDcWaiter
External

SubsidyRecoveryInitStWaiter
External

SubsidyRecoveryInitAccept
Standard: send message to executor (flow)

SubsidyRecoveryExecPromise
Standard: send message to initiator (flow)

SubsidyRecoveryExecT10Creator
Bridge SubsidyRepaymentInit

SubsidyRecoveryExecT10AcWaiter
External

SubsidyRecoveryExecExecute
Standard: do nothing

SubsidyRecoveryExecState
Standard: send message to initiator (flow)

SubsidyRecoveryExecAcRjWaiter
External

SubsidyRepaymentInitRequest
Bridge MailSender

SubsidyRepaymentInitPmDcWaiter
External

SubsidyRepaymentInitStWaiter
External

SubsidyRepaymentInitCheckRepayment
Standard: Listing E.10

SubsidyRepaymentInitAccept
Bridge MailSender

SubsidyRepaymentInitReject
Bridge MailSender

SubsidyRepaymentInitAcRjWaiter
External

Listing E.10: Code for SubsidyRepaymentInitCheckRepayment

```
if RepaidAmount equals FinalSubsidyAmount
then RepaymentOk
else RepaymentNotOk
```

Workflow elements

MailSenderFlow
`Fig. E.2`

PayerFlow
`Fig. E.3`

SubsidyFlow
`Fig. E.4`

SubsidyGrantInitFlow
`basic init worflow (Fig. B.3)`

SubsidyGrantExecFlow
`Fig. E.5`

SubsidyFormalEvaluationInitFlow
`basic init worflow (Fig. B.3)`

SubsidyFormalEvaluationExecFlow
`basic exec worflow (Fig. B.3)`

SubsidySubstantiveEvaluationInitFlow
`basic init worflow (Fig. B.3)`

SubsidySubstantiveEvaluationExecFlow
`basic exec worflow (Fig. B.3)`

SubsidyObligationsComplianceInitFlow
`basic init worflow (Fig. B.3)`

SubsidyFinalAmountDeterminationExecFlow
`Fig. E.6`

SubsidyPaymentInitFlow
`basic init worflow (Fig. B.3)`

SubsidyPaymentExecFlow
`basic exec worflow (Fig. B.3)`

RandomCheckInitFlow
`Fig. E.7`

RandomCheckExecFlow
`Fig. E.8`

RandomCheckExecInspectionHandlerFlow
`Fig. E.9`

SubsidyInspectionInitFlow
`Fig. E.10`

SubsidyInspectionExecFlow
`basic exec worflow (Fig. B.3)`

SubsidyRecoveryInitFlow
`basic init worflow (Fig. B.3)`

SubsidyRecoveryExecFlow
`Fig. E.11`

SubsidyRepaymentInitFlow
`Fig. E.12`

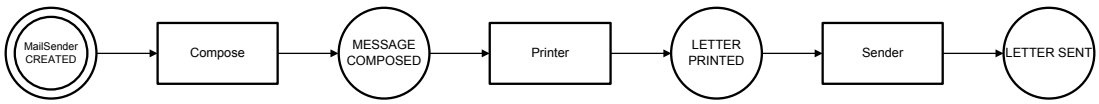


Figure E.2: MailSenderFlow

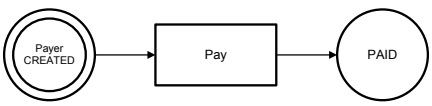


Figure E.3: PayerFlow

Figure E.4: SubsidyFlow

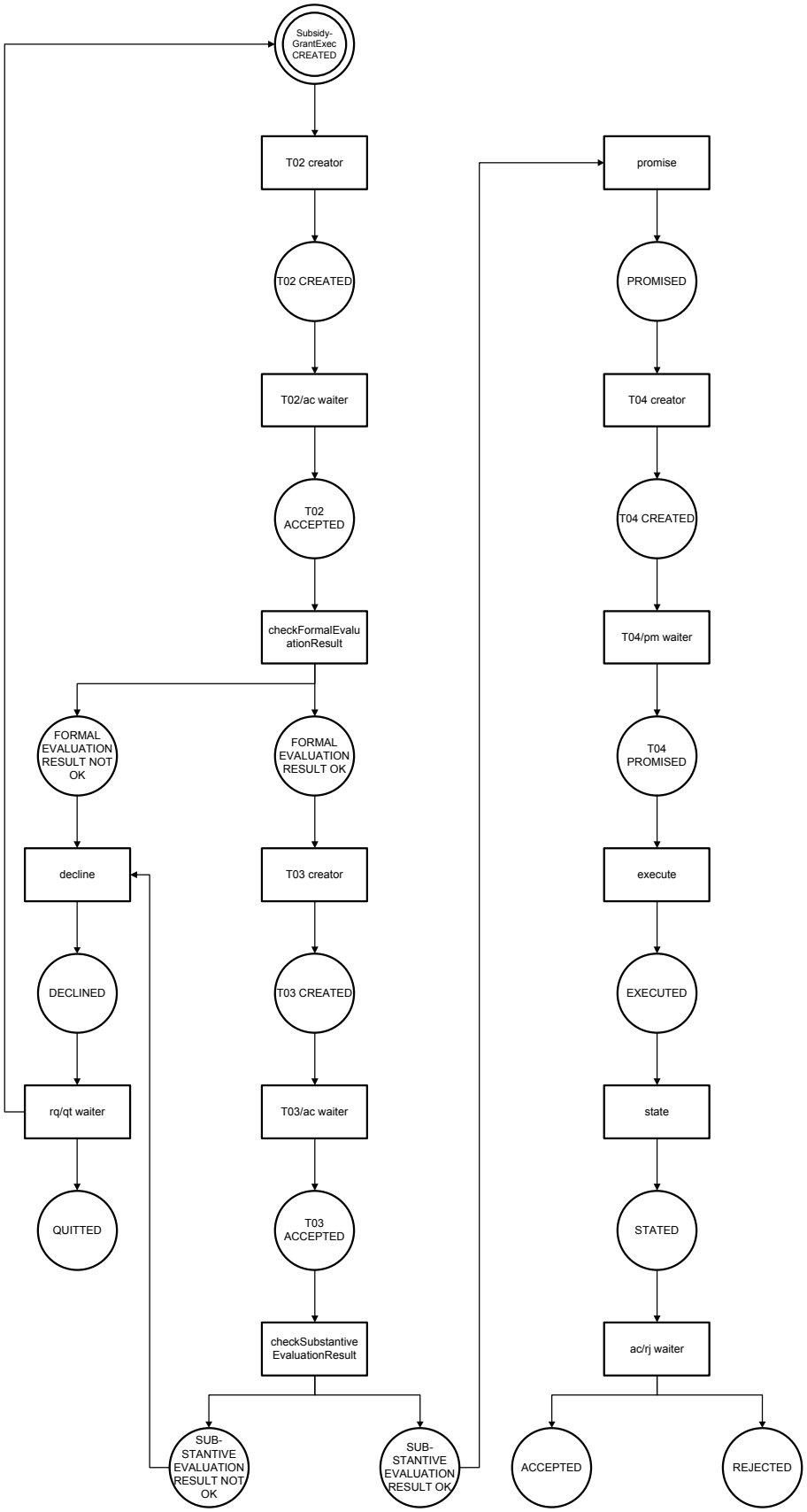


Figure E.5: SubsidyGrantExecFlow

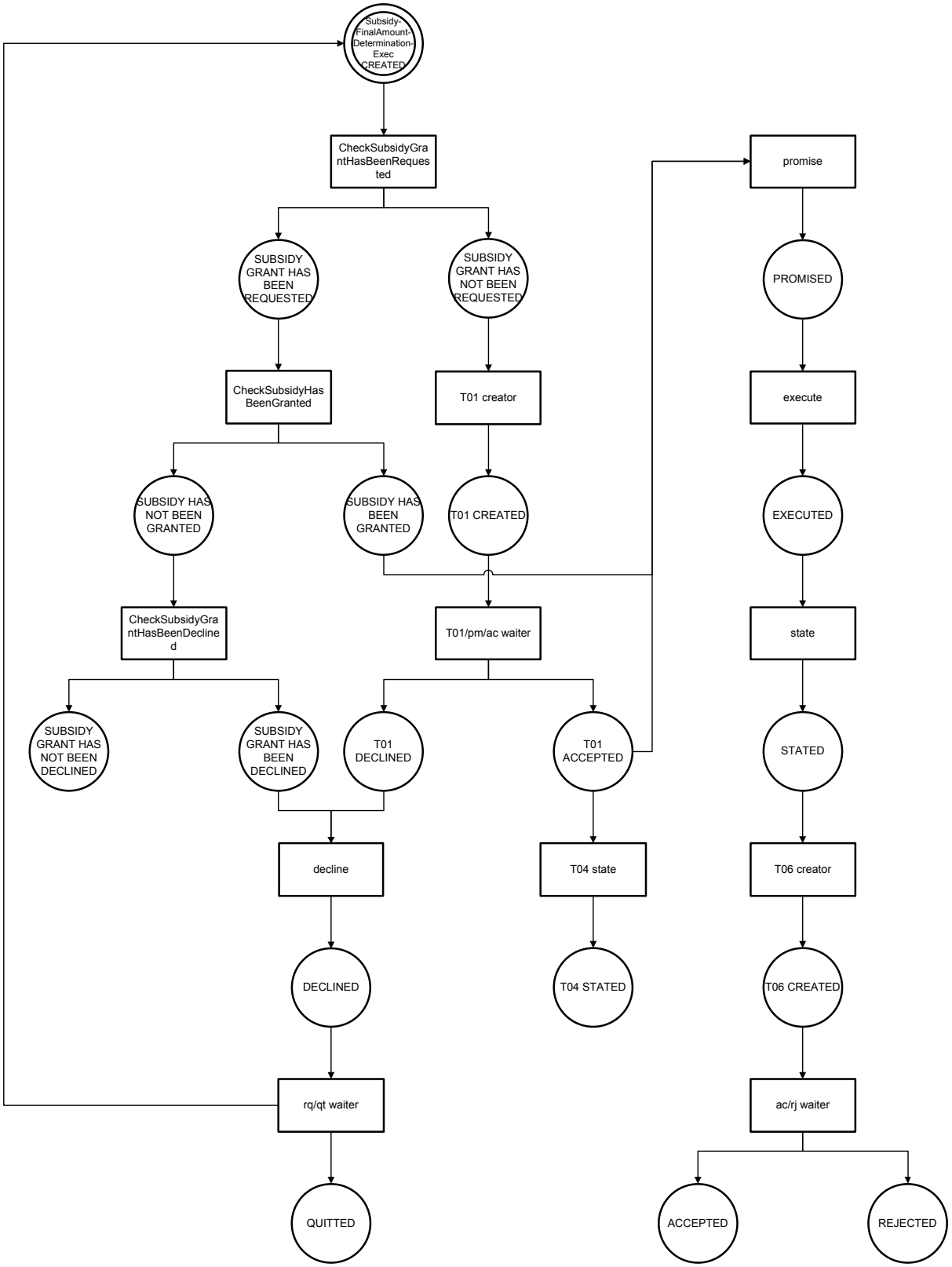


Figure E.6: SubsidyFinalAmountDeterminationExecFlow

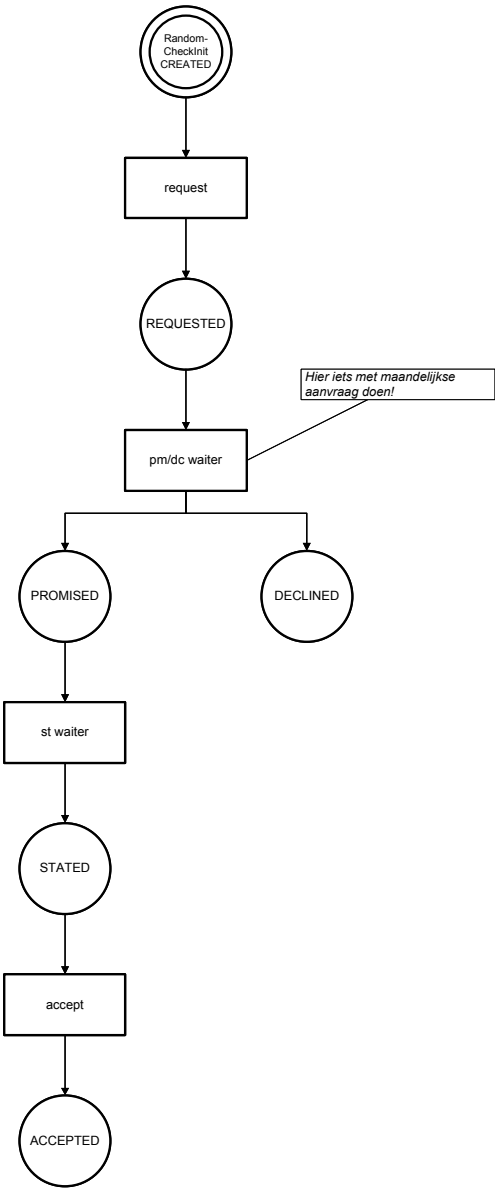


Figure E.7: RandomCheckInitFlow

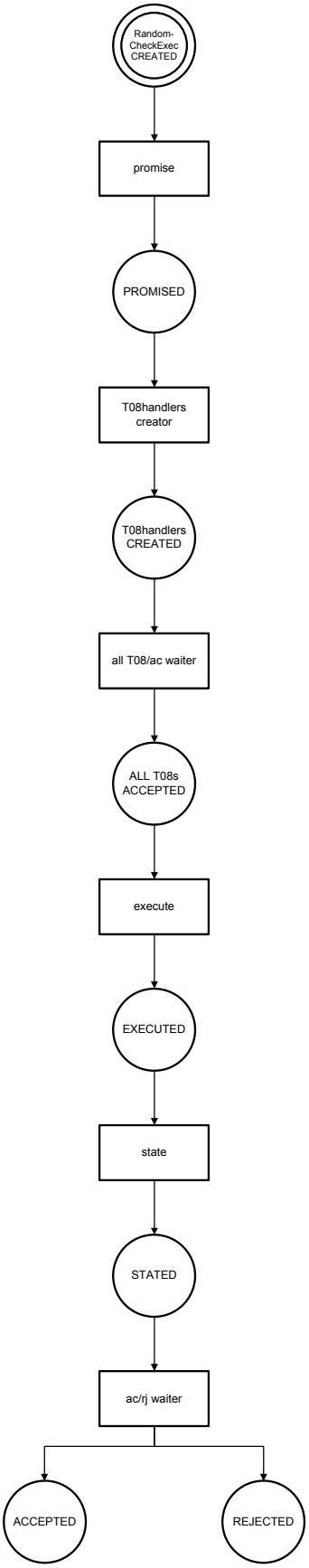


Figure E.8: RandomCheckExecFlow

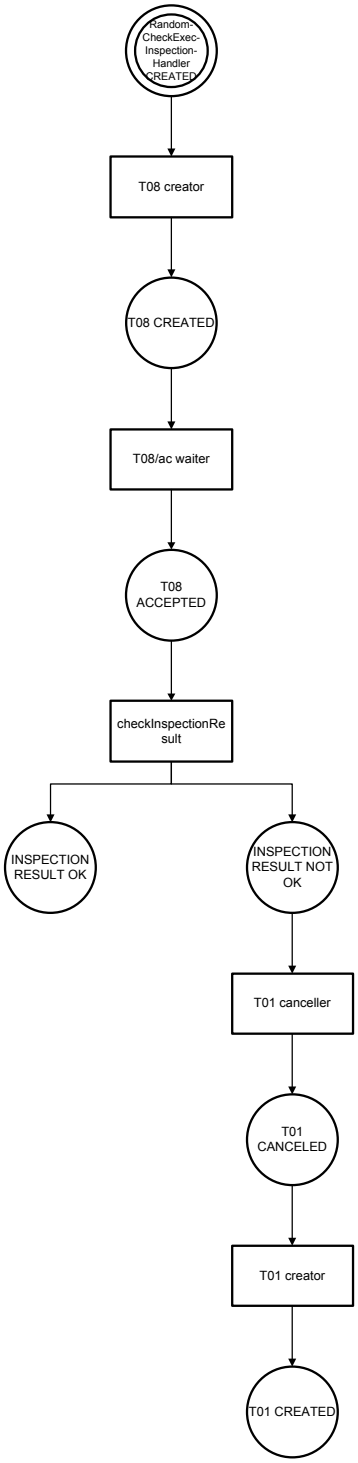


Figure E.9: RandomCheckExecInspectionHandlerFlow

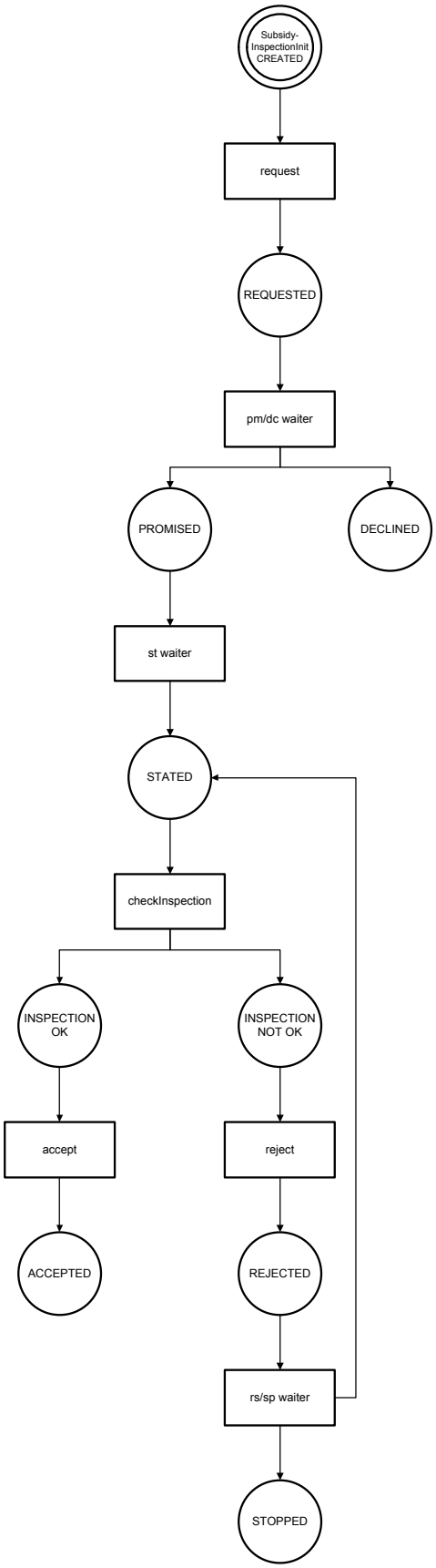


Figure E.10: SubsidyInspectionInitFlow

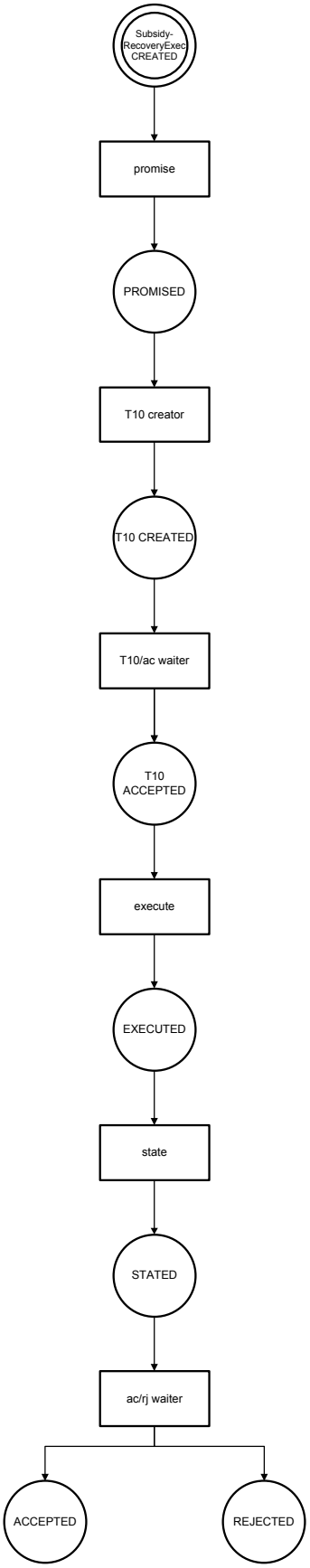


Figure E.11: SubsidyRecoveryExecFlow

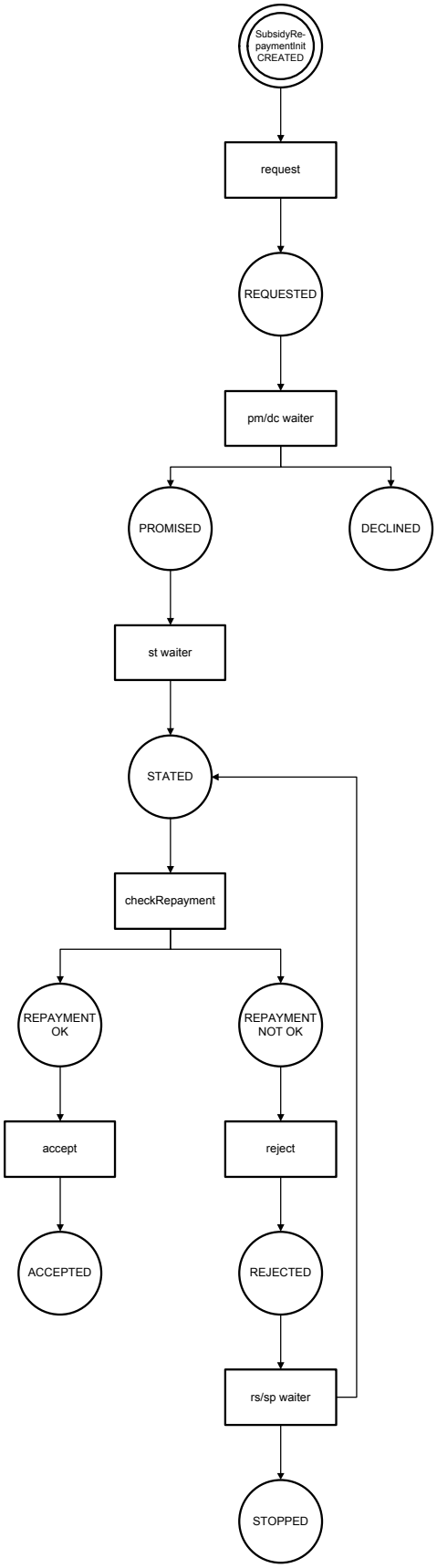


Figure E.12: SubsidyRepaymentInitFlow

F. NS EXPANDER MANUAL

Normalized Systems

ELEMENT EXPANDERS
DEVELOPERS GUIDE

Version 0.95

Table of Contents

1 Introduction

The *Normalized Systems Element Expanders*, in short *NS Expanders*, are a set of software programs, able to generate structured software elements in accordance with the Normalized Systems theory [1]. The NS Expanders assume an underlying 4-tier architecture based on the Java Enterprise Edition (JEE) standard. They are available for both Linux and Windows platforms, and support in general all major databases. Currently, they are only qualified and tested for the Jonas application server, and the presentation tier is only available for the Model View Controller implementation of the Cocoon XML publishing framework.

The NS Expanders distinguish the following modules or aggregations:

- An application, that can be deployed as an ear JEE enterprise archive.
- A component, that can be deployed as a jar EJB archive with corresponding war web tier archive and/or multiple service jar archives.
- The following types of NS elements:
  - Data elements
  - User connector elements
  - Flow elements
  - Task elements
  - Service connector elements

  that can be deployed within a component and made available in an application.

The NS Expanders will generate for the various types of modules, e.g. elements, components, and applications, not only the source code files, but also build files, configuration files, and provisioning scripts. We will now discuss these modules in more detail in the next sections.

2 Assembly Framework

The assembly framework expanders allow the developer to create skeletons of a new application or a new component.

2.1 Applications

The application skeleton is specified in an application descriptor file, for example `<newapp>.ad`:

```
/home/mannaert/cvs-palver/
newapp NewApplication
```

The first argument is the root directory of the cvs (Concurrent Versioning System) or source code tree. This argument needs to be specified in URI format, e.g. /D:/PALVER/current/ on a Windows platform. The arguments on the second line

identify the short name of the application, which will correspond to the name of the directory tree in the cvs-tree, and the full name of the application.

The application skeleton expansion can be invoked using the app_skel.sh script (app_skel.bat on Windows platforms), for instance:

./app_skel.sh <newapp>

2.2 Components

The component skeleton is specified in a component descriptor file, for example <newcomp>.cd:

```
/home/mannaert/cvs-palver/
newcomp newapp newelement-status
```

The first argument is the root directory of the cvs-tree. The arguments on the second line identify the name of the component, which will correspond to the name of the directory tree in the cvs-tree, and the short name of the application in which the component will be inserted. The final argument specifies the default URL that will be invoked for this component.

The component skeleton expansion can be invoked using the comp_skel.sh script, for instance:

./comp_skel.sh <newcomp>

Such a component can be added or integrated into one or more application builds using the comp2app.sh script:

./comp2app.sh <newcomp>

3 CRUDS Functionality

The expansion of software for standard CRUDS (Create-Retrieve-Update-Delete-Search) functionality of a data entity stored in a relational database, consists of the expansion of an instance of a data element and an instance of a user connector element.

3.1 Element Specification

3.1.1 Basic Attribute Description

The expansion of CRUDS functionality is specified in a data descriptor, for example `Purchase.dd`:

```
/home/mannaert/cvs-palver/
sitenet net.palver.uams Purchase tsms
String name ynn
String company yyn
String productCode yny10_20_30_40_50
String status yyyInitial_Processing_Finished
Integer amount ynn
Ln01net.palver.site.Site atSite nnn
Ln01net.palver.site.Technician orderedBy nnn
findByCompanyEq
findByCompanyEq_StatusEq
findByCompanyEq_AmountGt
findByCompanyEq_StatusEq_AtSiteEq
```

The first argument is the root directory of the cvs-tree in URI format. The arguments on the second line identify the name of the component (e.g. sitenet), the Java package name, the name of the actual data element, and the short name of the application in which the element will be inserted (e.g. tsms).

The lines 3, 4, 5, 6, 7, 8, and 9 specify the various fields or data attributes, and consist of three parts. The first part specifies the type of the data attribute. Possible types of attributes are the basic Java data types, like `Integer`, `Short`, `Long`, `String`, and `Date`. The use of longer textual descriptions in a text area is supported through the type `StringLong`. In case the attribute is a link to another data element, the type needs to start with `Ln`. In this case, three types of links are allowed:

1 Ln01: many-to-one link, the element software manages the relationship.
2 Ln02: many-to-one link, the JEE container manages the relationship (CMR).
3 Ln03: many-to-many link, the JEE container manage the relationship (CMR).
4 Ln04: one-to-many, reverse Ln02, the JEE container manages the relationship (Ln02 needs to be defined at the other side of the relationship).
5 Ln05: one-to-many, reverse Ln01, the element manages the relationship (Ln01 is needed, and the name of the Ln01 field needs to be identical to the name of the element where the corresponding Ln05 link is specified. Also the findBy<element>Eq finder needs to be specified in the data descriptor where the Ln01 relation is defined).
6 Ln06: many-to-many, reverse Ln06, the JEE container manages the relationship (Ln03 needs to be defined at the other side of the relationship).

The specification of the link type is followed by the package and class name of the other data element. In case the other data element belongs to another component, the package name is preceded with the name of the component archive, followed by a separation token: `<comp-name>.jar#`

The second part of the attribute specification is the name of the attribute, while the third part specifies three modifiers that can either be `y` or `n`. The first modifier tells the expander whether the attribute has to appear in the overview screens or not. If not, the attribute will only appear in detailed screens of individual instances of the data element. The second modifier specifies whether a finder method is required for that attribute, but is deprecated. The third modifier specifies whether the attribute can only take a limited set of possible values. If this is the case, an enumeration of possible values needs to be included; e.g. the values `10`, `20`, `30`, `40`, and `50` for the `productCode`. This set of possible values will be provisioned in the database, and can easily be extended later.

The lines `10`, `11`, `12`, and `13` specify the various search methods that need to be generated. The specification of such a method needs to start with the prefix `findBy`, and may contain any number of attributes. These attributes need to be separated with an underscore, and accompanied by a modifier. This modifier can currently take three possible values:

- `Eq`: equal to
- `Gt`: greater than
- `Lt`: less than

For instance, the third search method on line 12 will allow the user to search for instances of the Purchase data element where the company attribute is equal to a specific string, and the amount attribute is greater than a specific integer.

3.1.2 Advanced Element Features

3.1.2.1 Special Types

Standard stypes like String and Integer can have a substype in the type specification.

These subtypes can have for instance a specific meaning:

- `StringLink`: the string represents a link and will visualized accordingly.
- `StringMail`: the string represents an email address and will be visualized and handled accordingly.
- `StringFile`: the string represents a file, that will be uploaded to the server upon data entry.

Other subtypes can request for a more dedicated or advanced visualization:

- `IntSpinner`: will represent the integer field in a spinner widget.
- `LongSpinner`: will represent the long field in a spinner widget.
- `StringDate`: will represent the date field in a graphical calendar widget.

3.1.2.2 Special Fields

Several reserved field names will lead to the creation of a specific functionality related to that field.

- `enteredBy`: The system will automatically store the user that entered an instance of this data element. Users are not able to handle this field manually.
- `enteredAt`: The system will automatically store the time that an instance of this data element was entered. Users are not able to handle this field manually.
- `lastModifiedBy`: The system will automatically store the user that last modified an instance of this data element. No manual manipulation possible.
- `lastModifiedAt`: The system will automatically store the time that an instance of this data element was entered. No manual manipulation possible.
- `disabled`: In case an instance of this data element is deleted, the system will not remove it, but simply set this field to 'no', making the instance invisible for all standard operations. No manual manipulation possible.
- `parent`: The system will automatically create both an internal and a graphical tree representation for this element. A search method `findByParentEq` needs to be added to the element descriptor.
- `image`: The system will automatically generate a graphical representation for this data element. The NS Expanders will also generate an additional data element `<Element>Icon` that will permit the graphical positioning of icons of other data elements on this graphical representation.
- `name`: The system will make sure that this field has a unique value. In case no name field is specified, the NS Expanders will create an implicit name field, and create unique values for this field. In the latter case, no manual manipulation is possible.

3.1.2.3 Options

Various options can be configured, by adding keywords in the lower part of the data descriptor of the element.

- `optionCaching`: will activate a caching mechanism in the remote agent of the data element.
- `optionLookupField<field>`: will activate a multi-language mechanism, translating the field using a translation XSLT stylesheet.
- `optionStyleDriver<field>`: will activate a rendering mechanism, selecting the style of the rows in the overview tables based on the values of this field in the instances of the data element.
- `optionLinkDetail<field>`: will activate a retrieval mechanism, inserting the details of the link data instances into the details of the instances of this data element.
- `optionLinkDriver<field1>_<field2>`: will activate a interlink communication mechanism, retrieving the possible values of `field2` based on the selected value of `field1`.

3.2 Element Expansion

3.2.1 Invocation Scripts

The total expansion of CRUDS functionality may be invoked using the script `cruds_total.sh`, for instance:

```
./cruds_total.sh Purchase
```

This total expansion of CRUDS functionality can be divided in three parts. Every one of these parts is available through a separate script that will only invoke that part of the expansion:

- `./data_total.sh Purchase`: total expansion of the Purchase data element
- `./user_total.sh Purchase`: total expansion of the Purchase user connector element, comprising the second part of the CRUDS software
- `./user2app.sh Purchase`: insertion of the Purchase user element into the building and deployment of the application

3.2.2 Marginal Expansion

CRUDS functionality may be extended over time to add additional data attributes and/or search methods. For instance, consider the following data descriptor `Invoice.dd` that may be expanded before through the invocation of `cruds_total.sh`:

```
/home/mannaert/cvs-palver/
sitenet net.palver.uams Invoice tsms
String name ynn
String company yyn
String status yyyInitial_Sent_Booked_Paid
Integer accountingCode yny100_200_300_400_500
Ln01net.palver.uams.Purchase purchaseOrder nnn
findByCompanyEq
findByStatusEq
```

At a later point in time, an additional or marginal expansion may be performed using an additional data descriptor, for instance `Invoice_m.dd`:

```
/home/mannaert/cvs-palver/
sitenet net.palver.uams Invoice tsms
String reference ynn
Ln01account.jar#net.democritus.usr.Account customerAccount nnn
```

Once again, the first two lines specify generic information, while the following two lines specify additional data attributes for marginal expansion. Only new fields may be listed in de marginal expansion data descriptor and additional findBy search methods can only be defined for those new fields!

This marginal expansion can be invoked using a script `cruds_marg.sh`, for instance:

```
./cruds_marg.sh Invoice_m
```

This marginal expansion of CRUDS functionality can also be invoked for the individual elements through the following two scripts:

- `./data_marg.sh Invoice_m`: marginal expansion of the data element
- `./user_marg.sh Invoice_m`: marginal expansion of the user connector element

3.3 Element Functionality

3.3.1 Web User Screens

Deployed into an application, the CRUDS functionality of a data element such as `Purchase`, is available through web user screens. The main URI is the status overview `purchase-status`, or in general `<element>-status`. This gives you a page browsable overview of all instances of the data element, and access to several other CRUDS pages of the element:

- `<element>-entry`: the form to enter a new instance of the data element.
- `<element>-detail`: the details of a specific instance of the data element and the possibility to modify them.
- `<element>-search`: the various search methods that are available for the data element.

Based on the various options and special fields of the data element description, several other pages are made available:

- `<element>-tree`: the form generated tree structure in case the parent field is specified.
- `<element>-viewer`: the graphical representation in case the image field is specified..

Provisioning scripts are automatically generated to provision these URI's in the screens of the application. They are available in both Linux `.sh` and Windows `.bat` format, and both for an individual element or the entire component:

```
provision_<component>_screens
provision_<element>_screens
```

3.3.2 XML Data Access

Every instance of a data element is available in an XML export format through the URI `<element>-xmldetail`.

In case the special fields enteredAt, lastModifiedAt, and disabled are specified, the URI `<element>-change-partition-set` is made available. This will give an XML export where, based on a specific date, the following sets of instances of the data element are returned:

- The instances that have been updated since that date.
- The instances that have been added since that date.
- The instances that have been deleted since that date.
- The instances that have remained unchanged since that date.

3.3.3 Remote Java Agents

The basic CRUDS functionality is also available through the remote Java agents. Such an agent <Element>Agent can be called without knowledge of the remote communication. Functions offering standard CRUDS functionality include:

- `getInfoSet`
- `getFindSet`
- `getDetails`
- `modify`
- `create`
- `delete`

4 Workflow Processing

4.1 Element Specification

4.1.1 Target Flows

The expansion of a workflow state machine for a specific target data element is specified in a flow descriptor, for instance `Invoice.fd`:

```
/home/mannaert/cvs-palver/
sitenet net.palver.uams Invoicing bile tsms
net.palver.uams Invoice Status
InvoiceFlow
```

The first argument is the root directory of the cvs-tree. The arguments on the second line identify the name of the component, the Java package name, the full logical name of the workflow engine, the short name of the workflow engine archive, and the short name of the application in which the flow element will be inserted. The arguments on the third line specify the package name and the class name of the target data element, and the data attribute of that data element that will represent the state of the workflow state machine (cfr. Invoice.dd). The final argument represents the logical name of the workflow (and will need to be entered identically in the ConfigFlow Entry screen).

4.1.2 Action Tasks

The expansion of a task action operating on a specific target data element is specified in a task descriptor, for instance `InvoiceSender.td`:

```
/home/mannaert/cvs-palver/
sitenet net.palver.uams InvoiceSender tsms
net.palver.uams Invoice
InvoiceTask d
```

The first argument is the root directory of the cvs-tree. The arguments on the second line identify the name of the component, the Java package name, the full logical name of the task, and the short name of the application in which the task element will be inserted. The arguments on the third line specify the package name and the class name of the target data element. The final arguments represent the group name of the tasks operating on that target data element, and the type of the task. In the example above, the type of the action is `'d'` or default.

The only other type of a task that is currently available is the `'b'` or bridge type. In this case, the action will create an instance of another data element, and the user has to specify the package name and class name of that other data element in the task descriptor, for instance in `InvoiceUnpaidNotifier.td`.

```
/home/mannaert/cvs-palver/
sitenet net.palver.uams InvoiceUnpaidNotifier tsms
net.palver.uams Invoice
InvoiceTask bnet.palver.uams.Notifier
```

4.2 Element Expansion

4.2.1 Target Flows

The total expansion of the target flow element may be invoked using the script `flow_total.sh`, for instance:

```
./flow_total.sh Invoice
```

This total expansion of workflow functionality can be divided into the expansion of the actual flow element, and the insertion of the flow element into an application. This application insertion is also available in a separate script:

```
./flow2app.sh Invoice
```

This will perform the insertion of the Invoice flow element into the building and deployment of the application.

4.2.2 Action Tasks

The total expansion of the task element may be invoked using the script `task_total.sh`, for instance:

./task_total.sh InvoiceSender

This total expansion of task functionality can be divided into the expansion of the actual task element, and the insertion of the flow element into an application. This application insertion is also available in a separate script:

./task.sh InvoiceSender

This will perform the insertion of the InvoiceSender task element into the building and deployment of the application.

4.3 Element Provisioning

Before flow and task elements can actually be executed, the various tasks and timers of the workflow have to be provisioned into the data elements that represent the configuration of the workflow.

- An instance of a `Workflow` data element needs to be created for every flow element. This element provides CRUDS functionality through URI's such as `workflow-status` and workflow-entry.
- An instance of a `StateTask` data element needs to be created for every task element. This element provides CRUDS functionality through URI's such as `stateTask-status` and stateTask-entry.
- An instance of a `TimerTask` data element needs to be created for every timer element. This element provides CRUDS functionality through URI's such as `timerTask-status` and timerTask-entry.

For every task element `<Task>`, an instance of a parameter data element `<Task>Params` needs to be provisioned through the corresponding web URI's. A provisioning script is also generated to do this (both for Linux and Windows):

```
provision_<task-name>_parameters
```

For every flow element, the flow engine or driver needs to be provisioned into the database. This will allow the user to start the engine, stop the engine, and set the time interval of the engine. The script is available for both Linux and Windows:

```
provision_<flow-name>
```

5 References

[1] Mannaert H., Verelst J., "Normalized Systems – Recreating Information Technology Based on Laws for Software Evolvability", Koppa, Hasselt, Belgium, March 2009.

6 Appendix A: An Integrated Example

Part 1: Skeleton expansion

cd %EXPAND_ROOT%\components\expand\etc\test

edit newapp.ad and newcomp.cd

app_skel.bat newapp
comp_skel.bat newcomp
comp2app.bat newcomp

Part 2: Cruds functionality

Step 1: expand PDC data elements

cd %EXPAND_ROOT%\components\expand\etc\test

edit ARPA.dd, Bijlage.dd, Deelproduct.dd, etc.

cruds_total.bat ARPA
cruds_total.bat Bijlage
cruds_total.bat Deelproduct
cruds_total.bat Dienst
cruds_total.bat Doelgroep
cruds_total.bat Foutmelding
cruds_total.bat FinancieleMiddelen
cruds_total.bat HelpdeskMelding
cruds_total.bat Kernbegrip
cruds_total.bat Personeelsinzet
cruds_total.bat Persoon
cruds_total.bat Product
cruds_total.bat ServicePack

Step 2: build expanded source code

cd %DEMOCRITUS_ROOT%\applications\newapp
ant_install_all.bat

Step 3: populate newapp metadata for all basic components

start JOnAS

cd %DEMOCRITUS_ROOT%\components\utils\etc\init

populate_newapp.bat

cd %DEMOCRITUS_ROOT%\components\account\etc\init
populate_newapp.bat

cd %DEMOCRITUS_ROOT%\components\workflow\etc\init
populate_newapp.bat

Step 4: populate newcomp picklist values, refid's and user screens

cd %DEMOCRITUS_ROOT%\components\newcomp\etc\expand\scripts

choose the provisioning scripts you want to execute
provision_aRPA_list.bat
provision_aRPA_ref_id.bat
provision_aRPA_screen.bat
provision_bijlage_list.bat
provision_bijlage_screen.bat
provision_deelproduct_list.bat
provision_deelproduct_screen.bat
provision_dienst_list.bat
provision_dienst_screen.bat
provision_doelgroep_list.bat
provision_doelgroep_screen.bat
provision_financieleMiddelen_list.bat
provision_financieleMiddelen_screen.bat
provision_foutmelding_list.bat
provision_foutmelding_screen.bat
provision_helpdeskMelding_list.bat
provision_helpdeskMelding_screen.bat
provision_kernbegrip_list.bat
provision_kernbegrip_screen.bat
provision_personeelsinzet_list.bat
provision_personeelsinzet_screen.bat
provision_persoon_list.bat
provision_persoon_screen.bat
provision_product_list.bat
provision_product_screen.bat
provision_servicePack_list.bat
provision_servicePack_screen.bat

Step 5: test

http://localhost:8080/ or https://localhost:8443/

stop JOnAS

Part 3: Workflow Processing

Step 1.1: expand *basic* workflow data elements

cruds_total.bat ConfigFlow
cruds_total.bat ConfigTask
cruds_total.bat ConfigTimer

Step 1.2: build expanded workflow source code

cd %DEMOCRITUS_ROOT%\applications\newapp
ant_install_all.bat

Step 1.3: populate workflow picklist values, refid's and user screens

start JOnAS

cd %DEMOCRITUS_ROOT%\components\newcomp\etc\expand\scripts

provision_configFlow_list.bat
provision_configFlow_screen.bat

provision_configTimer_list.bat
provision_configTimer_screen.bat

provision_configTask_list.bat
provision_configTask_screen.bat

stop JOnAS

Step 2.1: marginal expansion of workflow 'status' field in Foutmelding element

marginal dd's may only contain new fields,
findBy methods can only be defined on new fields!

cruds_marg.bat Foutmelding_m

Step 2.2: alter database table for element Foutmelding

execute expanded SQL script to alter table and add additional column(s)
modify_pdc_foutmelding_.sql

remark: under Windows the expansion of modify_pdc_foutmelding_.sql does not yet work!
Could not generate modification sql script: java.io.FileNotFoundException:
C:\var\lib\postgres\modify_pdc_foutmelding_.sql
(The system cannot find the path specified)
→ temporary add the new columns using pgAdmin III

Step 2.3: build marginal expanded source code

cd %DEMOCRITUS_ROOT%\applications\newapp
ant_install_all.bat

Step 2.4: populate additional picklist entries for element Foutmelding

start JOnAS

provision_foutmelding_list.bat <-- NEW script to provision picklist entries

stop JOnAS

Step 3.1: expand new PDC workflow elements

edit Foutmelding.fd, FoutmeldingMailer.td

flow_total.bat Foutmelding
task_total.bat FoutmeldingMailer

Step 3.2: configure JOnAS service 'mail' and PDC service 'fout'

add the services 'mail' and 'fout' manually into %JONAS_BASE%\conf\jonas.properties:

jonas.services registry,jmx,jtm,**mail**,dbm,security,ejb,web,ear,**fout**
…
jonas.service.mail.factories MailSession1
…
last section in jonas.properties
jonas.service.fout.class be.provant.pdc.FoutafhandelingService

configure mail service in %JONAS_BASE%\conf\MailSession1.properties:

mail.from support@admin.provant.be
…
mail.host relay.pa.be

Step 3.3: develop task implementation class FoutmeldingMailerImpl.java

cd %DEMOCRITUS_ROOT%\components\newcomp\src\be\provant\pdc
implement FoutmeldingMailerImpl.java using the MailerBean session bean of component *utils*

Step 3.4: build expanded source code and task implementation class

cd %DEMOCRITUS_ROOT%\applications\newapp
ant_install_all.bat

Step 3.5: populate Foutmelding workflow picklist values and user screens

start JOnAS

provision_foutmeldingTaskStatus_list.bat
provision_foutmeldingTaskStatus_screen.bat

provision_foutmeldingTimerStatus_list.bat
provision_foutmeldingTimerStatus_screen.bat

provision_foutmeldingMailerParams_list.bat
provision_foutmeldingMailerParams_screen.bat

Step 3.6: populate FoutafhandelingService in workflow engine table

provision_foutafhandeling.bat
remark: this bat file is currently not expanded under Windows,
manually convert the unix shell script provision_foutafhandeling.sh into a bat script

step 4.1: GUI configuration of the *Foutmelding* workflow

cruds entry 'FoutmeldingFlow' ConfigFlow
(use the same name as specified on the last line of Foutmelding.fd)

cruds entry 'FoutmeldingMailer' configTask
(use the same name & processor name as defined on 2nd line of FoutmeldingMailer.td)

cruds entry 'default' FoutmeldingMailerParams
(use 'default' as name and choose be.provant.pdc.FoutmeldingMailerImpl from picklist)

step 4.2: test

http://localhost:8080/ or https://localhost:8443/

start 'fout' service via the engine control screen (e.g. https://localhost:8443/engine-detail)

monitor the workflow using the 'FoutmeldingTaskStatus Status' screen
(e.g. https://localhost:8443/foutmeldingTaskStatus-status)

search on the workflow using the 'FoutmeldingTaskStatus Search' screen
(e.g. https://localhost:8443/foutmeldingTaskStatus-search)

stop JonAS

Some example PDC descriptor files

newapp.ad

```
/D:/PALVER/current/
newapp pdcApplication
```

newcomp.cd

```
/D:/PALVER/current/
newcomp newapp
```

Bijlage.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc Bijlage newapp
String name yyn
String Url nnn
```

Dienst.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc Dienst newapp
String name yyn
```

Doelgroep.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc Doelgroep newapp
String name yyn
```

Foutmelding.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc Foutmelding newapp
String name yyn
StringLong omschrijving nnn
Ln01be.provant.pdc.Persoon melder yyn
```

HelpdeskMelding.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc HelpdeskMelding newapp
```

```
String name yyn
String Url nnn
```

Persoon.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc Persoon newapp
String name yyn
String emplId yyn
String telefoonnummer nnn
```

Product.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc Product newapp
String name yyn
StringLong omschrijving nnn
Ln03be.provant.pdc.Doelgroep doelgroepen nyn
String beschikbaarheid nnn
Ln01be.provant.pdc.Dienst verantwDienst yyn
Long reactietijdDagen nnn
Long reactietijdUren nnn
Long oplostijdDagen nnn
Long oplostijdUren nnn
Ln01be.provant.pdc.Persoon contactpersoon nnn
String prijs nnn
StringLong beperkingEnVoorwaarden nnn
Ln03be.provant.pdc.HelpdeskMelding helpdeskMelding nyn
Ln01be.provant.pdc.Product parent yyn
String isLeaf ynyYes_No
Ln03be.provant.pdc.Bijlage bijlage nyn
findByParentEq
findByIsLeafEq
findByPersoonEq
```

ServicePack.dd

```
/D:/PALVER/current/
newcomp be.provant.pdc ServicePack newapp
String name yyn
StringLong Omschrijving nnn
Ln03be.provant.pdc.Doelgroep doelgroepen nyn
String beschikbaarheid nnn
Ln01be.provant.pdc.Dienst verantwDienst yyn
Long reactietijdDagen nnn
Long reactietijdUren nnn
Long oplostijdDagen nnn
Long oplostijdUren nnn
Ln01be.provant.pdc.Persoon contactpersoon nnn
String prijs nnn
StringLong beperkingEnVoorwaarden nnn
Ln03be.provant.pdc.HelpdeskMelding helpdeskmelding nyn
Ln03be.provant.pdc.Product producten nyn
Ln03be.provant.pdc.Bijlage bijlage nyn
```

Foutmelding_m.dd

```
/C:/PALVER/expand-build-090909/
newcomp be.provant.pdc Foutmelding newapp
String status yyySubmitted_Notified_Accepted_Refused_Solved
String severity nnyHigh_Medium_Low
findByStatusEq
findBySeverityEq
```

ConfigFlow.dd

```
/D:/PALVER/current/
newcomp net.palver.conf ConfigFlow newapp
String name ynn
Ln01account.jar#net.democritus.usr.User responsible ynn
```

ConfigFlow.dd

```
/D:/PALVER/current/
newcomp net.palver.conf ConfigTask newapp
String name ynn
String packageName nnn
String processor ynn
String parameters ynn
String beginState nyn
String endState nnn
String failedState nnn
Ln01net.palver.conf.ConfigFlow flow yyn
findByBeginStateEq
findByFlowEq
```

ConfigTimer.dd

```
/D:/PALVER/current/
newcomp net.palver.conf ConfigTimer newapp
String name ynn
String packageName nnn
Long allowedPeriod ynn
String requiredAction ynyDoTask_AlterState_DoAndAlter
String processor ynn
String parameters ynn
String beginState nyn
String targetState nyn
String alteredState nnn
Ln01net.palver.conf.ConfigFlow flow yyn
findByBeginStateEq
findByFlowEq
```

Foutmelding.fd

```
/D:/PALVER/current/
newcomp be.provant.pdc Foutafhandeling fout newapp
be.provant.pdc Foutmelding Status
FoutmeldingFlow
```

FoutmeldingMailer.td

```
/D:/PALVER/current/
newcomp be.provant.pdc FoutmeldingMailer newapp
be.provant.pdc Foutmelding
FoutmeldingTask d
```

BIBLIOGRAPHY

[AEOtL00] B.R.T. Arnold, A.J.M. Engels, and M. Op 't Land, *FPS: een andere kijk op componenten en architectuur in de financiële wereld (deel 2)*, A & I (2000), no. 2, 24–32, dutch.

[Bar08] Joseph Barjis, *The Importance of Business Process Modeling in Software Systems Design*, Journal of The Science of Computer Programming **71** (2008), no. 1, 73–87.

[Boy07] Danah Boyd, *Information access in a networked world*, November 2007, Talk presented to Pearson Publishing, Palo Alto, California.

[Cap07] Capgemini Consulting, *Global CIO Survey 2007: Four Out of Ten CIOs at Multi-Billion Euro Enterprises Believe Their IT Function Cannot Deliver Enough Agility in an Environment of Accelerating Business Change*, March 2007, http://www.capgemini.com/news-and-events/news/cio_survey_2007/.

[DH04] Jan L.G. Dietz and Terry Halpin, *Using DEMO and ORM in Concert - a Case Study*, Advanced Topics in Database Research, Vol.3 (Keng Siau, ed.), vol. 3, Idea Group, 2004, pp. 218–236.

[dH09] Johan den Haan, *An Enterprise Ontology based approach to Model-Driven Engineering*, Master's thesis, Delft Univeristy of Technology, 2009.

[Die06] Jan. L. G. Dietz, *Enterprise Ontology*, Springer Berlin Heidelberg, 2006.

[Die08] _____, *Architecture: Building strategy into design*, Sdu Uitgevers bv, The Hague, The Netherlands, 2008.

[Die09a] _____, *DEMO-3 Models and Representations*, 2009, http://www.demo.nl/index.php?option=com_docman&task=doc_download&gid=121.

[Die09b] _____, *DEMO-3 Way of Working*, September 2009, http://www.demo.nl/index.php?option=com_docman&task=doc_download&gid=122.

[dJ09] Joop de Jong, *Integration Aspects between the B/I/D Organizations of the Enterprise*, Advances in Enterprise Engineering III (Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Antonia Albani, Joseph Barjis, and Jan L. G. Dietz, eds.), Lecture Notes in Business Information Processing, vol. 34, Springer Berlin Heidelberg, 2009, pp. 187–200.

[dJD10] Joop de Jong and Jan L.G. Dietz, *Understanding the realization of organizations*, Advances in Enterprise Engineering IV (Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Antonia Albani, and Jan L. G. Dietz, eds.), Lecture Notes in Business Information Processing, vol. 49, Springer Berlin Heidelberg, 2010, pp. 31–49.

[eK07] e Kenniscentrum, *Nederlandse Overheid Referentie Architectuur*, Tech. report, e Kenniscentrum, April 2007, http://e-overheid.nl/onderwerpen/architectuur-en-nora (dutch).

[Hal09] Terry Halpin, *Object-Role Modeling version 2*, Encyclopedia of Database Systems (Ling Liu and M. Tamer Özsu, eds.), Springer US, 2009, pp. 1941–1946.

[HBNV10a] Philip Huysmans, David Bellens, Dieter Van Nuffel, and Kris Ven, *Aligning the Constructs of Enterprise Ontology and Normalized Systems*, Sixth International CIAO! Workshop on the Fifth International Conference on Design Science Research in Information Systems and Technology (DESRIST 2010) (Aalst, Will and Mylopoulos, John and Sadeh, Norman M. and Shaw, Michael J. and Szyperski, Clemens and Albani, Antonia and Dietz, Jan L. G., ed.), Springer Berlin Heidelberg, June 2010.

[HBNV10b] _____, *Designing Enterprise Architectures based on Systems Theoretic Stability*, 5th International Conference on e-Business (ICE-B 2010) (Boris Shishkov David Marca and Marten Van Sinderen, eds.), SciTePress, Portugal, July 2010, pp. 157–162.

[HR06] Rick Hayes-Roth, *Two Theories of Process Design for Information Superiority: Smart Pull vs. Smart Push*, Command and Control Research and Technology Symposium: The State of the Art and the State of the Practice, June 2006.

[Kro] Marien R. Krouwel, *Data modeling from a business perspective*, research assignment.

[Leh80a] M.M. Lehman, *On understanding laws, evolution, and conservation in the large-program life cycle*, Journal of Systems and Software **1** (1980), 213–221.

[Leh80b] _____, *Programs, life cycles, and laws of software evolution*, Proceedings of the IEEE 68, 1980, pp. 1060–1076.

[LR01] M.M. Lehman and J.F. Ramil, *Rules and tools for software evolution planning and management*, Annals of Software Engineering **11** (2001), no. 1, 15–44.

[MF98] Jean-Philippe Martin-Flatin, *Push vs. Pull in Web-Based Network Management*, Tech. report, Swiss Federal Institute of Technology Lausanne, 1998.

[MV09] Herwig Mannaert and Jan Verelst, *Normalized systems: re-creating information technology based on laws for software evolvability*, Koppa, Kermt, Belgium, 2009.

[MvV09] Ruimtelijke Ordening en Milieubeheer Minister van Volkshuisvesting, *Tijdelijke sloopregeling*, May 2009, https://www.nationalesloopregeling.nl/downloads/Sloopregeling-MinisterieleRegelingvanMinisterievanVROM.pdf (dutch).

[NHBV10] Dieter Van Nuffel, Philip Huysmans, David Bellens, and Kris Ven, *Translating Ontological Business Transactions into Evolvable Information Systems*, 5th International-Conference on Software Engineering Advances (ICSEA 2010), August 2010.

[OtL08] Martin Op 't Land, *Applying architecture and ontology to the splitting and allying of enterprises*, Ph.D. thesis, Delft Univeristy of Technology, 2008.

[Rij] De Rijksoverheid, *Algemene wet bestuursrecht*, http://www.rijksoverheid.nl/onderwerpen/algemene-wet-bestuursrecht-awb.

[San09] Puspa I. Sandhyaduhita, *Extending the DEMO methodology for determining information systems requirements*, Master's thesis, Delft University of Technology, August 2009.

[SLSLW03] D. Simchi-Levi, E. Simchi-Levi, and M. Watson, *Tactical Planning for Reinventing the Supply Chain*, The Practice of Supply Chain Management (Terry P. Harrison, Hau L. Lee, and John J. Neale, eds.), Springer, 2003, pp. 13–21.

[Sni07] Brian Snijders, *Orchestration of (chain-) processes in subsidy agencies - developing a generic subsidy orchestration system*, Master's thesis, Delft University of Technology, October 2007.

[Sti] Stichting DEMO Kenniscentrum, *DEMO Knowledge Centre - Design & Engineering Methodology for Organizations*, http://www.demo.nl/.

[TV02] Nikos C. Tsourveloudis and Kimon P. Valavanis, *On the Measurement of Enterprise Agility*, Journal of Intelligent and Robotic Systems (2002), no. 33, 329–342.

[vdH10] Paulien van der Horst, *From business transactions to business processes workflow: Using DEMO and BPMN*, Master's thesis, Delft University of Technology, 2010.

[Wan09] Yan Wang, *Transformation of DEMO models into exchangeable format*, Master's thesis, Delft Univeristy of Technology, 2009.

ACRONYMS

ABD Actor Bank Diagram.

ADW Algemene Wet Bestuursrecht.

AM Action Model.

ARS Action Rule Specifications.

ATD Actor Transaction Diagram.

B-actor business actor.

BCT Bank Contents Table.

BSN Burger Service Nummer.

C-act coordination act.

C-fact coordination fact.

CM Construction Model.

CNG Compressed Natural Gas.

D-actor documental actor.

DEMO Design and Engineering Methodology for Organizations.

EJB Enterprise JavaBeans.

ERD ER Diagram.

ERM ER Modeling.

FDL Fact Definition List.

FM Fact Model.

GSDP Generic System Development Process.

I-actor intellectual actor.

IAM Interaction Model.

ICT Information and Communication Technology.

ISM Interstriction Model.

KvK Kamer van Koophandel.

LPG Liquid Petrol Gas.

MDE Model-driven Engineering.

NS Normalized System.

OCD Organization Construction Diagram.

ORM Object Role Modeling.

OS Object System.

P-act production act.

P-fact production fact.

PM Process Model.

PSD Process Structure Diagram.

RDW Rijksdienst Wegverkeer.

SSD State Space Diagram.

TPD Transaction Pattern Diagram.

TRT Transaction Result Table.

US Using System.

WOSL World Ontology Specification Language.