Enabling the automatic verification and exchange of DEMO models

Proefschrift ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college voor promoties
in het openbaar te verdedigen op

donderdag 31 maart 2022
om 12.30 uur precies

door

Mark Arnold Theodoor Mulder
geboren 24 november 1971
te Amersfoort

Promotor(en):
Prof. dr. H.A. Proper
Prof. dr. ir. J.L.G. Dietz (TU Delft)

Manuscriptcommissie:
Prof. dr. M.C.J.D. van Eekelen
Prof. dr. J. Barijs (San José State University, Verenigde Staten)
Prof. dr. J.M. Tribolet (Universidade Técnica de Lisboa, Portugal)
Dr. S.J.B.A. Hoppenbrouwers
Dr. D. Bork (Technische Universität Wien, Oostenrijk)

Enabling the automatic verification and exchange of DEMO models

Dissertation to obtain the degree of doctor
from Radboud University Nijmegen
on the authority of the Rector Magnificus prof. dr. J.H.J.M. van Krieken,
according to the decision of the Doctorate Board
to be defended in public on

Thursday, March 31, 2022
at 12.30 pm

by

Mark Arnold Theodoor Mulder
born on November 24, 1971
in Amersfoort (the Netherlands)

Supervisor(s):
Prof. dr. H.A. Proper
Prof. dr. ir. J.L.G. Dietz (TU Delft)

Manuscript Committee:
Prof. dr. M.C.D.J. van Eekelen
Prof. dr. J. Barijs (San José State University, USA)
Prof. dr. J.M. Tribolet (Technical University of Lisbon, Portugal)
Dr. S.J.B.A. Hoppenbrouwers
Dr. B. Bork (TU Wien, Austria)

About the image[1] on the cover

The column located at the highest point of the hill (number 23) is left blank to represent the concept of "emptiness" (Sunyata), which is the key theme in the Heart Su¯tra. The Heart Su¯tra is a popular sutra in Maha¯y¯ana Buddhism. Its Sanskrit title can be translated as "The Heart of the Perfection of Wisdom".

The sutra famously states, "Form is empty, emptiness is form.". It is a condensed expose on the Buddhist Mahayana teaching of the Two Truths doctrine, which says that ultimately all phenomena are sunyata, empty of an unchanging essence. This emptiness is a 'characteristic' of all phenomena, and not a transcendent reality, but also "empty" of an essence of its own. Specifically, it is a response to Sarvastivada teachings that "phenomena" or its constituents are real[2].

[1] https://commons.wikimedia.org/wiki/File:Wisdom_Path,_(Hong_Kong).jpg
[2] https://en.wikipedia.org/wiki/Heart_Sutra

Abstract

The Design and Engineering Methodology for Organisations (DEMO) is a core method within the discipline of Enterprise Engineering (EE). It enables the creation of so-called *essential* models of enterprises. Such models are enterprise models that aim to focus on the organisational essence of an enterprise by leaving out (as much as possible) details of the socio-technical implementation. The organisational essence is then expressed primarily in terms of the actor roles involved, and the business transactions between these roles.

The DEMO method has a firm theoretical foundation. At the same time, there is an increasing uptake of DEMO in practice. This also results in a need for enterprise-grade tool support for the use of the method.

In this thesis, we report on a study concerning the selection, configuration, and extension, of an enterprise-grade tool to support the use of DEMO in practice with the need of the automatic verification and exchange of DEMO models. The configuration of the selected tool framework to support DEMO modelling, provided general insights regarding the development of enterprise-grade tool support for (model-driven) methods such as DEMO, while also providing feedback on the consistency and completeness of the DEMO Specification Language (DEMOSL); the specification language that accompanies the DEMO method.

Foreword

This thesis is the result of a 8-year lasting research to gain the maximum grade obtainable by study, in the Netherlands. For me it was the most challenging task on a logical subject ever. Combining this with changing jobs, starting my own company, next to matters concerning my family; wife and 3 kids was not an easy road to take. Compared to all of the above, the thesis was simple.

Over these 8 years one specification and seven papers have been published as an abstract of this thesis and associated artefacts. These papers are listed in chapter Publications on page 21.

I got on the path of obtaining PhD via my father-in-law Ed ten Voorde, who kept telling me it would be a wise thing to do. So finally in 2012 my research idea started with automating the then popular term of omnichannel communication. In my opinion this had to be based on DEMO, which in turn needed more specification for it to be automated. It turned out to be more than only more specification which in turn resulted in the title of this thesis. Via Martin Op't Land, Raymond Slot and Wiebe Wiersema this idea was formalized in 2013. During the last 8-years a lot of people embarked in my journey towards today: Jan Dietz, Hans Mulder, Jan Verelst, Herwig Mannaert, Erik Proper, José Tribolet, David Aveiro, Robert Pergl, Dominik Bork, Joseph Barjis, Stijn Hoppenbrouwers, Johan Versendaal, Steven van Kervel, Holke Visser, Ronald Lans and Fiodor Bodnar, all of whom I'd like to thank for their lasting support. Thank you!! I'm fairly sure I failed to mention everyone that helped me, but be sure my thanks concern you too.

ir. Mark A.T. Mulder

Contents

Acronyms

ACD	Actor role Competence Diagram
AFD	Actor role Function Diagram
ALD	Application Layer Diagram
ANTLR	ANother Tool for Language Recognition
AM	Action Model
API	Application Programming Interface
AR	Action Research
ARK	Action Rules Kind
ARS	Action Rules Specification
ARD	Action Rules Diagram
AT	Attribute Type
ATK	Aggregate Transaction Kind
BCT	Bank Contents Table
BLD	Business Layer Diagram
BMP	BitMaP image file
BPM	Business Process Modelling
BPMN	Business Process Model and Notation
BRM	Business Rule Management
CAR	Composite Actor Role
CM	Construction Model
DEMO	Design and Engineering Methodology for Organisations
DEMOSL	DEMO Specification Language
DFD	Data Flow Diagram
DSR	Design Science Research
DTD	Document Type Definition
EBNF	Extended Backus - Naur Form
EAR	Elementary Actor Role
EE	Enterprise Engineering
EEi	Enterprise Engineering institute
EEWC	Enterprise Engineering Working Conference
ECF	Elementary Construction Flaw
ERD	Entity Relation Diagram
ERP	Enterprise Resource Planning
ET	Entity Type
ETK	Elementary Transaction Kind

EO	Enterprise Ontology
FBM	Fact Based Modelling
FC	Flowchart
FK	Fact Kind
FM	Fact Model
FQDN	fully qualified domain name
GCMF	General Conceptual Modelling Framework
GDPR	General Data Protection Regulation
GSK	General Step Kinds
GUID	Global Unique IDentfier
ICT	Information and Communication Technology
IAM	Identity Access Management
IDEF	Integration Definition
IFK	Independent P-Fact Kind
ISO	International Organization for Standardization
IUT	Information Use Table
MU	Model Universe
NEN	NEderlandse Norm
OCD	Organisation Construction Diagram
OER	Organisational Essence Revealing
OFD	Object Fact Diagram
OHD	Organisation Hierarchy Diagram
ORM	Object Role Modeling
PM	Process Model
PK	Product Kind
PNG	Portable Network Graphics
PPTX	PowerPoint XML Presentation (Microsoft)
PSD	Process Structure Diagram
PSI-theory	Performance in Social Interaction theory
PSK	Process Step Kind
QR	Quantitive Research
SEA	Sparx Enterprise Architect
SIB	Scope of Interest Boundary
SRD	Security Role Diagram
SoI	Scope of Interest
SVG	Scalable Vector Graphics
ROME	Return On Modeling Effort
RPSK	Revoke Pattern Step Kind
RSK	Revoke Step Kind
TEoO	The Essence of Organisation
TLD	Technology Layer Diagram
TK	Transaction Kind
TPD	Transaction Pattern Diagram
TPT	Transaction Product Table

TKSK	Transaction Kind Step Kind
TPSK	Transaction Process Step Kind
UI	User Interface
UML	Unified Modelling Language
VBA	Visual Basic for Applications
VISI	'Voorwaarden scheppen voor de Invoering van Standaardisatie ICT in de bouw'
VSDX	Microsoft Visio XML Drawing
WIS	Work Instruction Specification
WOSL	World Ontology Specification Language
XML	Extensible Markup Language
XMI	XML Metadata Interchange
XSD	XML Schema Definition
XOER	eXtended Organisational Essence and Revealing

Publications

During this research we published intermediate results in the following papers and conferences:

Enterprise Engineering Instituut:

[1] - J. L. G. Dietz and M. A. T. Mulder. *DEMO Specification Language 3.7*. 2017

EEWC 2018, Enterprise Engineering Working Conference:

[2] - M. A. T. Mulder. 'Validating the DEMO Specification Language'. In: *Enterprise Engineering Working Conference*. Springer, 2018, pp. 131–143

FBM 2018, On the Move to Meaningful Internet Systems, Fact Based Modelling:

[3] - M. A. T. Mulder. 'Towards a Complete Metamodel for DEMO CM'. in: *OTM Confederated International Conferences' On the Move to Meaningful Internet Systems'*. Springer, 2018, pp. 97–106

EEWC 2019, Enterprise Engineering Working Conference:

[4] - M. A. T. Mulder. 'A design evaluation of an extension to the DEMO methodology'. In: *Advances in Enterprise Engineering X*. Advances in Enterprise Engineering XIII. Springer, 2019, pp. 55–65

EEWC 2020, Enterprise Engineering Working Conference:

[5] - Mark.A.T. Mulder and Henderik.A. Proper. 'Evolving the DEMO Specification Language'. In: *Enterprise Engineering Working Conference*. 2020

PoEM 2020, International Federation for Information Processing WG 8.1 working conference on the Practice of Enterprise Modelling:

[6] - Mark.A.T. Mulder and Henderik.A. Proper. 'Towards Enterprise-Grade Tool Support for DEMO'. in: *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer. Nov. 2020, pp. 90–105. DOI: 10.1007/978-3-030-63479-7_7

CAiSE 2021, International Conference on Advanced Information Systems Engineering:

[7] - Mark.A.T. Mulder and Henderik.A. Proper. 'On the Development of Enterprise-Grade Tool Support for the DEMO Method'. In: *CAiSE 2021*. 2021

SoSym 2021, Special Section on PoEM 2020:

[8] - Mark.A.T. Mulder and Henderik.A. Proper. 'On Enterprise-Grade Tool Support for DEMO'. in: *Journal Software and Systems Modeling*. 2021

Paper Trail

As part of the research conducted towards this Thesis, several papers have been authored and published. Below, we provide a mapping showing how each of these papers contributes to the content of this thesis.

M. A. T. Mulder. 'Validating the DEMO Specification Language'. In: *Enterprise Engineering Working Conference*. Springer, 2018, pp. 131–143

From paper section	To thesis
1 Introduction	section 1.2.2
1.1 Aspect models	section 2.1.3
1.2 Problem statement	section 1.4.1
1.3 Observations	section 3.2
1.4 Research Design	section 1.5
2 Notion of metamodel	section 2.5
3 DEMOSL inconsistencies	section 3.3
3.1 Metamodel	section 3.3.1
3.2 Construction Model	section 3.3.2
3.3 Process Model	section 3.3.3
3.4 Action Model	section 3.3.5
3.5 Fact Model	section 3.3.4
4 Conclusions and future research	sections 7.2, 8.1 and 8.2

Table 1: Paper [2] trail

M. A. T. Mulder. 'Towards a Complete Metamodel for DEMO CM'. in: *OTM Confederated International Conferences' On the Move to Meaningful Internet Systems'*. Springer, 2018, pp. 97–106

From paper section	To thesis
1 Introduction	sections 1.2.2 and 2.1.3
2 Research Design	section 1.5
3 Extending DEMOSL	sections 2.5 and 4.2.1
3.1 Extending the verification rules	section 4.1.2
3.2 DEMO Exchange Model	section 4.1.4
4 Extending DEMOSL for the CM	chapter 4
4.1 Transaction Kind	section 4.2.5
4.2 Aggregate Transaction Kind	section 4.2.6
4.3 Elementary Actor Role	section 4.2.7
4.4 Composite Actor Role	section 4.2.8
5 Conclusions and future research	sections 7.2, 8.1 and 8.2

Table 2: Paper [3] trail

M. A. T. Mulder. 'A design evaluation of an extension to the DEMO methodology'. In: *Advances in Enterprise Engineering X*. Advances in Enterprise Engineering XIII. Springer, 2019, pp. 55–65

From paper section	To thesis
1 Introduction	sections 1.2.2 and 6.9
2 Research Method	sections 1.5 and 6.2
3 Model iterations	section 6.1
3.1 Iteration A	section 6.2
3.2 Iteration B	section 6.3
3.3 Iteration C	section 6.4
3.4 Iteration D	section 6.5
3.5 Iteration E	section 6.6
4 Extending OER	section 6.7
5 Conclusions and future research	sections 7.2, 8.1 and 8.2

Table 3: Paper [4] trail

Mark.A.T. Mulder and Henderik.A. Proper. 'Evolving the DEMO Specification Language'. In: *Enterprise Engineering Working Conference.* 2020

From paper section	To thesis
1 Introduction	section 1.2.2
2 Research background	sections 1.5 and 5.1.1
3 MU theory	section 2.3
4 Perspectives on DEMO models	section 2.2
5 Overview of the metamodels	chapter 4
5.1 High Level Ontological metamodel	sections 2.6 and 4.1.1
5.2 Ontological metamodel	sections 2.6 and 4.1.1
5.3 Data Exchange metamodel	sections 2.6 and 4.1.4
5.4 Visualisation metamodel	sections 2.6, 4.1.3, 4.2.2, 4.2.3, 4.2.8, 4.3, 4.3.2, 4.3.3, 4.4.1, 4.4.2, 4.5, 4.5.1 and 4.5.2
5.5 Visualisation Exchange metamodel	sections 2.6 and 4.1.4.2
5.6 Models and exchange models	–
6 Reflection	sections 7.1 and 8.1
7 Future Research	section 8.2
8 Conclusion	section 7.2

Table 4: Paper [5] trail

Mark.A.T. Mulder and Henderik.A. Proper. 'Towards Enterprise-Grade Tool Support for DEMO'. in: *IFIP Working Conference on The Practice of Enterprise Modeling.* Springer. Nov. 2020, pp. 90–105. DOI: `10.1007/978-3-030-63479-7_7`

From paper section	To thesis
1 Introduction	section 1.2.2
2 DEMO	sections 1.2.2, 2.1.1 and 2.1.3
3 The DEMO Meta-Model	chapter 4
4 Requirements for Tool Selection	section 5.1.1
5 Considered Tools	section 5.1.2
6 Implementation	sections 5.2, 5.2.1, 5.2.3, 5.2.6, 5.3.1, 5.3.2 and 5.3.5
7 Cases	sections 6.1 to 6.6
8 Conclusions and Future Research	sections 7.2 and 8.2

Table 5: Paper [6] trail

Mark.A.T. Mulder and Henderik.A. Proper. 'On the Development of Enterprise-Grade Tool Support for the DEMO Method'. In: *CAiSE 2021*. 2021

From paper section	To thesis
1 Introduction	sections 1.2.2 and 1.5.4
2 Enterprise-Grade Tool Support	sections 1.5, 2.8 and 5.1.1
3 DEMO Background	sections 1.2.2, 2.1.1 and 2.1.3
4 Meta-Model Extensions	sections 2.2, 3.3.2, 4.2.8.1, 4.5.3, 4.6 and 5.2.5
5 Visualisation	sections 3.3.6, 4.1.3 and 6.8
6 Model Exchange	section 4.1.4
7 Conclusions and Future Research	sections 7.2 and 8.2

Table 6: Paper [7] trail

Mark.A.T. Mulder and Henderik.A. Proper. 'On Enterprise-Grade Tool Support for DEMO'. in: *Journal Software and Systems Modeling*. 2021

From paper section	To thesis
1 Introduction	sections 1.2.2 and 1.4.1
2 Research Methodology	section 1.5
3 Enterprise-Grade Tool Support	section 2.8
4 DEMO	sections 1.2.2, 2.1.1 and 2.1.3
5 The DEMO Meta-Model	chapter 4 and section 4.1.1
6 Tool Selection Requirements	section 5.1.1
7 Considered Tools	section 5.1.2
8 Implementation	sections 5.2, 5.2.1, 5.2.3, 5.2.6, 5.3, 5.3.2 and 5.4
9 Summary of Experiences	section 7.2.5
10 Cases	sections 6.1 to 6.6
11 Conclusions and Future Research	sections 7.2 and 8.2

Table 7: Paper [8] trail

Chapter 1

Introduction

1.1 Background

This thesis is concerned with the Design and Engineering Methodology for Organisations (DEMO) methodology, and more particularly how its ways of thinking, modelling, and working are connected to the way of supporting (fig. 1.1) [9]. The way of modelling is about the notation of the models, the way of working is about the path from reality towards the model. Together they are called a methodology and the methodology is contained in the way of thinking. In DEMO the way of supporting is neglected. The automated support of modelling is becoming an essential need for business analysts. Models of the organisation of large enterprises are beyond the capability of the human brain. Ensuring the internal correctness of the models by verification is no longer a feasible task for a project team. Moreover, the exchange of these models in any, digital or physical, document is too complex for anyone. Therefore, we did research on this subject to make the automated support of modelling, verification and exchange of DEMO models possible. Business analysts, whether or not using the DEMO methodology, should use all these ways in their daily practice.

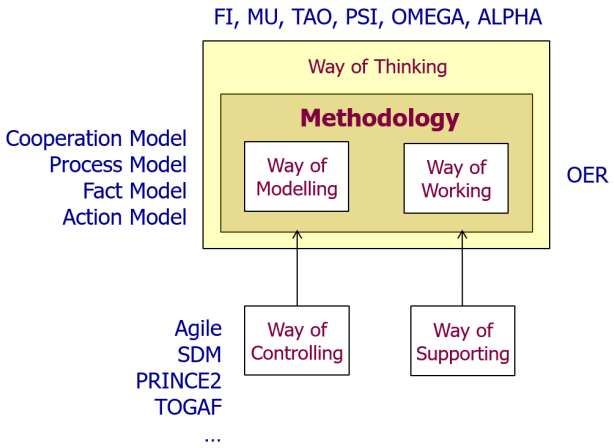


Figure 1.1: Five Ways of the DEMO methodology

When conducting a business analysis, analysts need to document all knowledge domains of the organisation that stakeholders need to know about, i.e. the overall business needs [46]. Modelling involves abstracting from reality, while focusing on those elements that are relevant to the purpose [10, 11]. It is nearly impossible to model all the knowledge domains of an organisation. It is only possible to model some of them and describe others vaguely at most. Therefore, when we try to model an organisation from a holistic (as possible) point of view, we use only the parts of the organisation that can be described and formally modelled. Words can describe knowledge, but a description in an analysable model is preferred. Although AI has already come a long way towards the interpretation of plain texts and big data, writers and readers may still understand the semantics of information in different ways [47]. When modellers create a model of this information, they act as interpreters of the organisational information. By definition, modelling languages limit interpretations of modelled parts; the resulting model can, therefore, have an unambiguous interpretation. Every knowledge domain has its own characteristics and, therefore, needs its own suitable modelling method (e.g. one cannot model data knowledge using process modelling methods and vice versa). Consequently, we need multiple modelling methods or ontologies to model the knowledge domains of an organisation.
As we do a walk-through of an organisation, we will see multiple knowledge domains (e.g. processes; data; business rules; applications; interfaces; infrastructure; flows). Business Process Modelling (BPM) claims to capture business processes [48]; Fact Based Modelling (FBM) claims to capture data [49]; ArchiMate claims to capture the functional side of businesses, applications, and infrastructure [12]; Business Rule Management (BRM) claims to capture functional requirements [50]; and the Design and Engineering Methodology for Organisations (DEMO) claims to capture the essence of business processes, data, and business rules [25, 9]. These knowledge domains jointly describe the same organisation. All models of each knowledge domain should have connections [51] to adjacent domains. Unfortunately, no single model covers all knowledge domains as each only focuses on its own domain.
So far, the promising integrated modelling solution has been Enterprise Engineering (EE) [52, 53, 54, 55, 56] with its DEMO methodology. This scientific methodology puts, among other things, process aspects, data structure, and business rules together to describe an organisation coherently and comprehensively in the sense of 'as holistically as possible'.

1.2 The DEMO Methodology

1.2.1 Viewpoints on an Organisation

An enterprise can gain correct insight into its operational processes by modelling the way it has been organised. The more complex this organisation, the greater the need for models of the organisation [57]. The acquired insights into the organisation can be about various knowledge domains and, therefore, need multiple viewpoints on that organisation. Regrettably, process models often only use representation of process diagrams from an implementation viewpoint and, therefore, do not reduce complexity. When business ana-

lysts complete process diagrams, they subsequently repeat the modelling effort for other processes resulting in a collection of independent, but possibly related, process diagrams of the organisation. These process diagrams, then, do not enhance the comprehensibility of the overall processes as one is not able to grasp the whole viewpoint of the process model at once.

Next to this process modelling exercise, a data viewpoint can be used to collect the data structures of the organisation. Organisations tend to reduce data modelling activities to data structures that need to be automated. Business modellers usually skip the data models of the business as the need for data models is most manifest in the automation field. The collected data models are stored separately from each other, are neither compared nor aligned, and there is no internal drive to achieve completeness of this viewpoint in the organisation. In the end, the data model consists of various fragmented, partial models.

Another step in obtaining insight is gathering business rules from the organisation. Business analysts often obtain these rules in the context of an application. Once analysts have determined the boundaries of an application, the business rules for data entry and internal workflow will either be described in a design document or will be programmed directly. The business rules, written as application rules, are disconnected from the previously defined processes or data structures and can rarely be found in the description of these models. Due to the disconnected models, the organisation implements several partial applications that in total cover every functional aspect of the organisation and some applications that cover multiple functional parts within the organisation. The organisation might have or not have any documentation of the present applications, but this documentation will be likely scattered around the organisation and potentially lack any coherence between the descriptions. Subsequently, this kind of scattered low-detailed application documentation can be collected but that does not mean it enables one to achieve insight into the organisation.

Maintaining an as holistic as possible overview of what one has modelled also needs complexity reduction which should be an intrinsic property of a viewpoint. Although models can be used to reduce complexity, they often only reduce complexity on a specific viewpoint and increase complexity by adding new concepts. People working in an organisation may often perceive the number of final concepts and corresponding explanations in a model to be more complex than the operational organisation [58]. This number of concepts likely reduces usability and increases the chance of necessary recreation of the models when needed in the future. Therefore, one needs to use a coherent modelling methodology with a minimal number of concepts.

Moreover, to have and maintain the overview of a model leads to the formulation of practical requirements this model should meet. First, modelling a business process as a whole requires a modelling methodology that can reduce complexity to such an extent that one can divide the model into sections that can be comprehended and recognised by individuals within the organisation. One should do this sectional modelling without losing the connection to the complete model of the organisation [59]. Furthermore, with regard to data, one needs to model the structure in such a way that allows the complete model to remain preserved while one can show partial models in diagrams. Subsequently, the information must be comprehensively structured from the viewpoint of business processes

themselves. The model of an organisation needs a single set of all elements, relations, and viewpoints, e.g. diagrams, allowing for references to this repository of the created model viewpoints. An internally sound and structured metamodel of this repository will lead to insightful models.
However, current modelling methods and diagram techniques mainly aim at partially modelling unique viewpoints of the organisation. Let us look at some examples:

a) BPM is widely used to describe the implemented process of an organisation. It is particularly useful in describing performed process steps of individuals or the kinds of jobs within the organisation or department. The description language and drawing technique of BPM are capable of describing the functional perspective of an organisation [60].
b) Data models, such as Unified Modelling Language (UML) class diagrams, are used to describe the information structure of organisations. These diagrams are capable of capturing the structure of the information from an object-oriented viewpoint. When these data are stored in a relational database, the data models are often described in an Entity Relation Diagram (ERD). These diagrams illustrate the use of a modelling technique depending on the target solution or implementation instead of a business problem or situation [61].
c) A BRM application captures all business rules to be able to describe them all. This way of working leads to rules without any link to a process or a data structure.

These examples show that each model has its own strength and has to be combined with other models to give the required insights into the organisation.

1.2.2 DEMO and its Development

The DEMO [25, 9] method is a core method (based on a theoretically founded method*ology*) within the discipline of EE [54]. The DEMO methodology focuses on the creation of so-called *essential* models of enterprises. The latter kind of models aim to capture the organisational essence of an enterprise by leaving out (as much as possible) details of the socio-technical implementation. The organisational essence is then expressed primarily in terms of the actor roles involved, as well as the business transactions [13] (and ultimately in terms of speech acts [62]) between these actor roles. More specifically, an essential model comprises the integrated whole of four aspect models: the Construction Model (CM), the Action Model (AM), the Process Model (PM) and the Fact Model (FM). Each aspect model involves its own kinds of diagrams and tables, and one or more cross-model tables, providing viewpoints on the complete model. The aspect models overlap in elements; specifically, the concept of *transaction kind* appears in all aspect models; therefore, being the de-facto integration point of the model as a whole. The methodology is discussed in more detail in section 2.1.
DEMO has strong methodological, and theoretical, roots [13, 25, 54]. At the same time, there is an increasing uptake of DEMO in practice as well. The latter is illustrated by the active usage (and certification) community[1], reported cases concerning the use of

[1] `http://www.ee-institute.org/en`
`https://www.linkedin.com/company/enterprise-engineering-institute`

DEMO [14, 15, 9], as well as the integration with other mainstream enterprise modelling approaches such as ArchiMate [16, 51] and BPMN [17, 63, 64, 18]. Meanwhile, DEMO has gained a proven track record in process (re)design, software specifications based on the organisation, modelling business rules and proving compliance to, e.g. the General Data Protection Regulation (GDPR) [65] and other International Organization for Standardization (ISO) and NEderlandse Norm (NEN) norms [19]. Additionally, an early descendant of DEMO, called 'Voorwaarden scheppen voor de Invoering van Standaardisatie ICT in de bouw' (VISI), has evolved into the ISO 29481-2:2012[2] (BIM related) standard for the construction sector.
The DEMO Specification Language (DEMOSL) defines the accompanying integrated modelling environment. As such, it should allow for the integration of the different DEMO models used in organisations when applying DEMO, while also enabling tool support, visualisation, as well as the exchange of models. The studied version of DEMO is called DEMO-3. Its specification language is published in [1].
DEMO [66] is currently Enterprise Engineering's key methodology. In this thesis we examine version 3 of DEMO. DEMO is a method modellers use to capture an organisation from four viewpoints, or aspect models. First, the **CM** models how subjects, with actor roles, in the organisation communicate with each other through standard communication patterns, called transactions. These diagrams show the dependencies between roles in execution and in information. The high abstraction level makes this a compact diagram in relation to the implementation of the organisation. The **PM** models show the relations between the process steps of interrelated transactions. This is used to explain the dependencies between transactions together with partial business rules. The **FM** describes information structures and the ways in which this information is created by the products of transactions. This model is often compared to a data model although it has only a partial match with that type of model. Finally, completing the model, the **AM** models business rules and work instructions concerning data structures and transactions. Per non-trivial process step minimal one specification shows the input and conditions to proceed in the transaction pattern or advance to other transactions. This specification is used to model all details of the organisation. The integrated model comprises these four aspect models and together they represent the essence of an organisation.
The DEMO methodology claims that the description of the organisation will comply with five properties. All aspect models have related properties that make the model a *coherent* integrated whole. The transactions of the CM connect the PM and the FM. The transaction steps of the AM connect to the PM and the FM from the detailed business rules. Next, DEMO clearly defines all model components in a *consistent* way throughout the whole model. Therefore, the whole DEMO model is claimed to be consistent [9]. Furthermore, the methodology is *comprehensive*, as it has enough components to fully describe the essence of the organisation. Despite its comprehensiveness, the methodology is still *concise* as it has just enough components that describe the organisation. Finally, Dietz designed DEMO to describe the *essential* parts of the organisation. These parts are likely to be stable in the future of the organisation.

[2] `https://www.iso.org/obp/ui/#iso:std:iso:29481:-2:ed-1:v1:en`

The history of DEMO is broader than Enterprise Engineering (EE). The development of DEMO took place between 1990 and 2004 [67]. Its inception can be traced back to 1992 when prior work on the conceptual modelling of Information Systems was transferred to that of the organization as a social system. "The development of DEMO is an advancement of the SMARTIE project, of which the overall goal was to 'search for sound theoretical foundations for the discipline of Information Systems Engineering'. The period from 1990 to 1994 is marked by the theoretical struggle to combine the formal and mechanistic SMARTIE model with Habermas' theory of communicative action."[13]
From its very inception, the DEMO methodology has been applied to various domains. Its application started with the conceptual modelling of information systems (1990) and the use evolved through social systems, system engineering, business process redesign and modelling, to the modelling of business processes and ontology combined with object role modelling (2004).
By 2004 DEMO was *"... characterized as positivist, constructive conceptual development, as the resulting blueprint of an organization is independent of the analyst. This is only possible within a realist ontological position, which DEMO bases upon from the systemic theory of Bunge. Although DEMO does not question its means of application, it easily fits within a technocratic management tradition that regards an organization as the means to achieve corporate goals. DEMO's abstraction from human beings to rational actors dismisses any social complications that may arise within an organisation. Accordingly, it primarily regards organization as a problem of coordination."* [67] Although DEMO is a useful methodology, the *"... research indicates, practitioners apply a combination of various methods, techniques, and perhaps methodologies to tackle problems."* The demand for extensibility and interoperability was already present in 2004.
Analysts have modelled in DEMO for about 25 years now for smaller and larger organisations. While most cases are proprietary, over a dozen reports on the practical use and result of DEMO are provided by the Enterprise Engineering institute (EEi) website "http://www.ee-institute.org". To manually model larger organisations (i.e. drawing by hand or with a simple drawing tool) is becoming difficult. It is almost impossible to verify the model, keep partial models consistent, apply all syntactic and semantic rules, and keep track of all details. This complexity needs tool-supported DEMO modelling. Therefore, we need an internally sound method and metamodel, because it is complicated to automate non-consistent methods, notations and techniques. The automation effort, the DEMO tool, needs a metamodel to be able to contain, verify, and communicate the DEMO model. As a result of automation, a verified model could be executed in a workflow engine, primarily but not exclusively limited to a DEMO workflow engine [39]. Moreover, the verified model can be used to feed follow-up modelling efforts in data and business rule areas [63]. In this thesis we use the terms of validation and verification from the IEEE standard 24765 [20] which defines verification as "confirmation, through the provision of objective evidence, that specified requirements have been fulfilled" and validation as "confirmation, through the provision of objective evidence, that the requirements *for a specific intended use* or application have been fulfilled". Therefore, to allow for the creation and direct or indirect execution of DEMO models, one needs models that are validated (but that is beyond the scope of this thesis) and verified, containing all aspect

models, enhancing the usability of the methodology.

1.2.3 Knowledge Base

In the period 1994–2019, most publications on the subject of DEMO involved papers in conference proceedings, journal articles, book chapters, books, and theses. The philosophical base was finished in 1996 [68]. This work led to the first version of DEMO [13]. Thereafter, several books on DEMO were published in 2006 [25] and in 2013 [26]. Although DEMO was developed at the University of Delft as we mentioned earlier, other universities have joined the effort (Madeira, Prague, Saint Hellen, and the University of Pretoria). Currently, many master students and PhD students are involved in DEMO research or associated parts of it. Starting from the DEMO 2 book [25], the World Ontology Specification Language (WOSL) was introduced and followed by a more detailed specification of the DEMO ontological metamodel in 2015 named DEMOSL 3.1. This specification has received updates in the years that followed. This research focusses on the 2017 DEMO version 3.7 as in [1], [69], [26] and also takes into account definitions and explanations from DEMO version 2 [25].
DEMO has proven to be a suitable method for modelling the construction of an organisation. Various research activities (e.g. [21]) have verified the usability of the DEMO method itself.
The description of the DEMO method is used in various PhD theses like Hommes [30], Terlouw [70, p31-38], Kervel [39, p65-78], Reijswoud and Dietz [13, p79-82], Mulder [21, p46-58], Jong [71, p23-32], and Ettema [72, p122-124]. An extensive summary of the DEMO theory is presented in section 2.1.

1.2.4 Practice

In daily EE practice, experts use DEMO aspect models to get insight into the complexity of organisations. Modelling often starts with the CM and PM because most business managers think in terms of processes. The modelling is then extended with the FM to show the connection with data structures (the area of information analysts and programmers) and to reveal the data knowledge domain. Modellers seem to rarely use the AM in practice because of the vast amount of work associated with it. The readability and expressiveness of the text version of the AM is low for a large number of stakeholders. Therefore, this aspect of modelling is executed very late in the process or not at all.
Also, the available toolset appears insufficiently capable of modelling large organisations; the current tools do not comply with the specifications of DEMOSL 3.7 [1] and they do not support the practice of modelling adequately enough.
Automated support for building an organisation model comes with some requirements. Teams working on a model using a tool to create the model while complying with its specification require a central repository containing all model components. Moreover, the verification implementation is needed to check the repository model against all verification rules. More requirements come to mind when choosing such a technical environment. All these aspects together express a need for a tool that can support the DEMO method.

1.3 Research Focus

We have searched for a single metamodel for all notations at a single point in time. This search conjectured not only that this model does not exist, but also that not all methods have a complete metamodel themselves. Hommes [30] started a metamodel for some aspect models but did not finish this work. As a result, there is no publicly available, and complete, metamodel for DEMO. When investigating DEMO, we also found that the ontological metamodel does not cover the complete modelling, is not entirely coherent between aspect models and is inconsistent in some ways [2]. This finding led to the question whether a complete, coherent, concise, consistent, and essential metamodel could be created. These claims should then be traceable to the aspect models and metamodel of DEMO. Therefore, completing the DEMO metamodel is a necessary step towards this overall metamodel.

In using DEMO to model an organisation, we found some shortcomings, mistakes, and other mishaps in the documentation and usage of the methodology. This research wants to focus on the practical usage of DEMO and solve the shortcomings from a scientific standpoint to be able to use this methodology in more significant depth to model organisations. To fix these shortcomings, we have to validate the current models and metamodels to find the gaps in their methods and notation. By validating and expanding the methodology, notation, and metamodel of DEMO, we can use the methodology to model, exchange, and execute business processes. To be able to take all these steps, we need to have a formal definition of the notation, syntax, and semantics of DEMO. We can use formality for enterprise modelling aspects as being intersubjectively understandable and machine processable [22]. This will reduce its inconsistencies and make DEMO an even stronger methodology.

In conclusion, this research has focussed on creating this metamodel together with the specification rules that come with the model. It also includes the usage of the metamodel in practice and contains a fully implemented metamodel that can be used to exchange the DEMO models of an organisation. A few references to partial metamodels have been made in this thesis to attempt to describe the exchange models or rules syntax. This research has tested all presented models on usability in automated tools. The metamodel is complete for the examples that have been used to create it. During this research, we built a new DEMO tool as a verification and validation tool for checking the models on the DEMOSL specification and for communicating the model with the organisation.

The combination of the complete methodology and the automated support of DEMO could lead to a higher acceptance of the methodology in practice. The scope of this thesis is too small to measure this change, but we have faith that the use of this methodology will change the future of organisation modelling.

1.4 Research Objectives

1.4.1 Problem Statement

The focus of the research will be on the completeness, and correctness of the formalisation of DEMO 3 models in DEMOSL 3.7 to accomplish automation of the verification, and exchange of DEMO models using a automation tool. The research aims to make DEMO usable to model organisations, in such a way that the models are complete, consistent, verifiable, and comprehensible for stakeholders. When business analysts use the DEMO methodology in practice, i.e. to model complex organisations, modelling tools are essential to create a model within an acceptable time frame with an acceptable level of quality. When one is creating a tool to support a methodology, the metamodels of all aspect models and diagrams have to be aligned correctly to create the overall metamodel. Automation means unambiguous logic and must, therefore, perfectly match aspect models. When using the DEMO aspect models without tools, or with separate tools for a few aspect models, one is, e.g. not likely to spot incoherence or the definition variances that obstruct integration of the aspect models. Therefore, when developing such tool support, it is also important to ensure that the respective metamodels of all aspect models, as well as the associated diagrams and tables, are aligned in order to create an integrated and consistent metamodel. Automated tool support also requires an unambiguous logic and must also accommodate the different aspect models. Therefore, selecting a tool or tool base (on which one builds a DEMO specific tool part) that can support a DEMO model is not straightforward. Even more, developing a modelling tool based on a method's metamodel will also disclose possible limitations of the method and vice versa.
To be able to research integrated models, and in assessing the validity of the two artefacts (DEMOSL and associated tool support), driving the different design science iterations [23], we did not limit ourselves to fictitious examples but specifically also included (see section 6.1) real-world situations to enhance the modelling capability of a method. We have decided to use organisations in need of an organisation model to conduct our research. Therefore, the research will be invasive and results further in the process will be influenced by the previous results. The Design Science Research (DSR) approach as described by Hevner et al. [73], Aken and Andriessen [74], and Wieringa [24] are methods that support this type of research, as will be explained in section 1.5.

1.4.2 Main Research Question

The main research question of this thesis is:

> How can we formalise the DEMO metamodel in such a way that automated tools can assist modelling in DEMO and that DEMO models can be logically verified and exchanged for practical usage?

Below, we explain this research question in more detail.

How can we formalise...	Formalising the metamodel is necessary to contain all existing DEMO models. A formalised metamodel serves as a guide for modellers to restrict their design freedom in such a way that the semantics of a model are conceived equally amongst modellers.
...the DEMO metamodel...	The current metamodel covers only the high-level ontological level; we have to extend it to cover the ontological metamodel level. Also, there appears to be a need to extend the metamodel to represent more notions of an organisation in the DEMO model.
...in such a way that automated tools can assist modelling in DEMO...	Automation helps the modeller when the model complexity exceeds cognitive capabilities. Completeness of the model is better when it is machine checked.
...and that DEMO models can be logically verified...	First predicate logic is a way to reason about the models.
...and exchanged for practical usage	Exchanging models with other tool components like simulation, execution, translation, and possible other targets is needed to optimise methodology usage. Since we are not doing fundamental research, modelling has to have practical relevance.

Although the representation of the models is part of the DEMOSL metamodel, we decided to define the representation explicitly in this thesis. As will be explained in section 2.5 a metamodel should facilitate the model exchange and that can be automated or the exchange of a visual representation of the model.

1.4.3 Research Questions

The books *Enterprise Ontology* [25], the more recently *The Essence of Organisation* [26], and the *Enterprise Ontology 2* [9] describe the DEMO methodology. This methodology enables modellers to model the essence of an organisation integrally in terms of a number of aspect models. We want to help the professionals with the creation of DEMO models. We aim to to that with the creation of the tool to simplify modelling and, therefore,

Our **research objective** is to
formalise the metamodel of DEMO
in such a way that
DEMO models can be verified automatically and
DEMO models can be exchanged.

To attain this result, we have to answer the following detailed research questions:

- How can we create the metamodel in such a way that we can implement it in a tool?
  This is important partly because the ontological level of the current metamodel is too high to be useful. e.g. Implementation of 'Action Rule' as a single entity without any property types is less practical;
- How can we verify a DEMO model in such a way that it is beneficial, and not a limiting factor, for modelling an organisation?
  This is important because applying all rules of a DEMO model can be too restrictive to benefit the modelling process. e.g. applying all rules of a DEMO model at once requires the modeller to fill all attribute types and property types of a transaction kind. Sometimes only high level information is available when modelling starts. Being restricted by all rules would suffocate the modelling process;
- How can we use the DEMO model and the respective metamodel for automating the modelled organisation or parts of it?
  This is important because only a formal correct metamodel can be a base for follow-up tooling.

See chapter 3 for a further extensive explanation of the reasons why the metamodel should be extended.

1.5 Research Method

This thesis describes the formalisation of the metamodel of DEMO which can be seen as an artefact in the sense of design science. The aim of this research project is to build this artefact while maintaining its integrity. In this context, approaches such as Quantitive Research (QR), exploratory research and explanatory research are less useful. QR, with the aid of questionnaires, gives us insight into details of the perceived meaning of the artefact and the way people either use or do not use it. That does not bring us any closer to the validation of the core of the DEMO metamodel artefact. In other words, in this research project we are not interested in the way people use the DEMO methodology. Moreover, exploratory research could explain the DEMO design but would not be able to improve it. Those aspects of explanatory research that are useful can also be found in DSR. In the next section we will explain which DSR is most relevant to this research project.

1.5.1 Design Science Research

Design Science Research is the embodiment of closely related cycles of activities to design engineering artefacts [73]. The cycles and usage of these cycles are different depending on the method one uses. Three mainstream methods within DSR could be relevant to our research aim: Hevner et al. [73], Aken and Andriessen [74], and Wieringa [24]. In what follows we will describe them and explain why we choose the application of Wieringa's approach to Design Science Research in our own research.

1.5.2 DSR according to Hevner et al.

The design science research paradigm [73] seeks to support the design and construction of useful artefacts that can solve problems expressed as functional requirements. To solve those problems, one needs to gain knowledge and understanding of the whole problem domain where the specific problem resides as well as the solution domain in which the artefact resides.
Design Science is aimed at conducting, evaluating, and presenting engineering artefacts. Hevner et al. argue that the empirical sciences and the behavioural sciences are also of relevance here. The design of an enterprise as a social system is also an engineering artefact. Additionally, Hevner's methodology contains guidelines to construct an artefact and ways to assess its qualities as a solution for a problem.

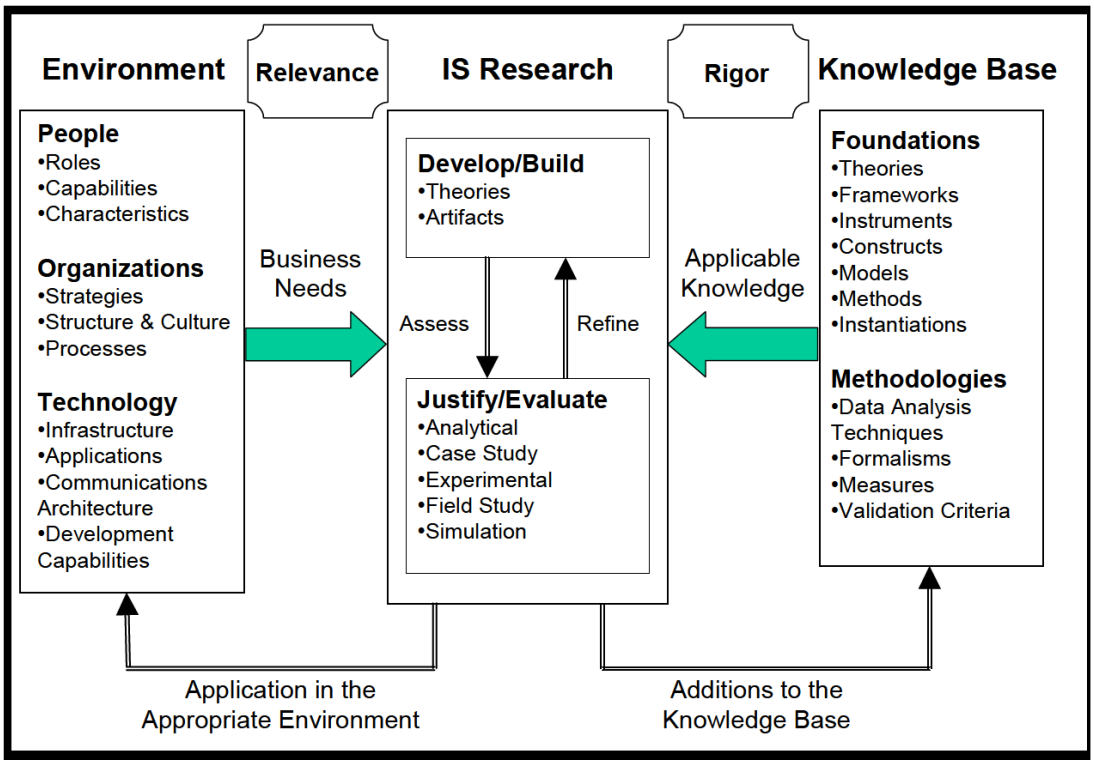


Figure 1.2: Hevner DSR [73]

1.5.2.1 Design Science Cycles

Hevner et al. describe a three-cycle view with a design cycle, a rigour cycle, and a relevance cycle. These cycles may be iteratively repeated several times to improve the solution to a particular engineering problem.

1. The Design Cycle involves the design and construction of an artefact, followed by an assessment of its function at an early stage. This assessment provides feedback for incremental modifications and enhancements of the artefact. Alternative design

options can be investigated and rejected or accepted. This process continues until an acceptable result is achieved.

2. The Rigour Cycle addresses the need for and identification of the use of proper scientific foundations and formal methods for the construction of the artefact. This rigour cycle allows for a distinction between science and (best) practices and is based on scientific foundations.

3. The Relevance Cycle in DSR is an empirical assessment of the artefact and its environment. This assessment involves meeting the validation of the requirements and objectives of this research through field testing.

1.5.2.2 Design Science Guidelines

Design Science guidelines are necessary when creating an artefact. Design as an artefact means that any engineering artefact is a designed and purposeful solution to a real and relevant problem. After the creation of an artefact one must evaluate its design to assess its qualities as a solution to this problem. Research in Design Science should provide clear theoretical and methodological contributions to generally address problems and also provide specific solutions to specific problems. The artefact must be constructed using rigorous formal methods and it should be created from available means. There may be non-deterministic processes at work. opposed to the deterministic calculation of a solution. Finally, the results of DSR have to be suitably presented to various audiences and stakeholders.

1.5.2.3 Design Science Artefacts

DSR distinguishes four types of artefacts. The first type are constructs that consist of vocabulary, symbols, and grammars. The second type are models as abstraction and representation of the real world using these constructs. Thirdly, artefacts can be methods including algorithms and processes. Finally, DSR is also about instantiation of artefacts such as prototypes and implementations.

1.5.3 DSR according to Aken and Andriessen

Aken and Andriessen characterise DSR as research on field problems with the emphasis on the production of prescriptive knowledge. DSR provides the solution to these field problems while the justification of the solution resides in its pragmatic validity. This kind of research is aimed at answering 'how' as well as 'what' questions [74].

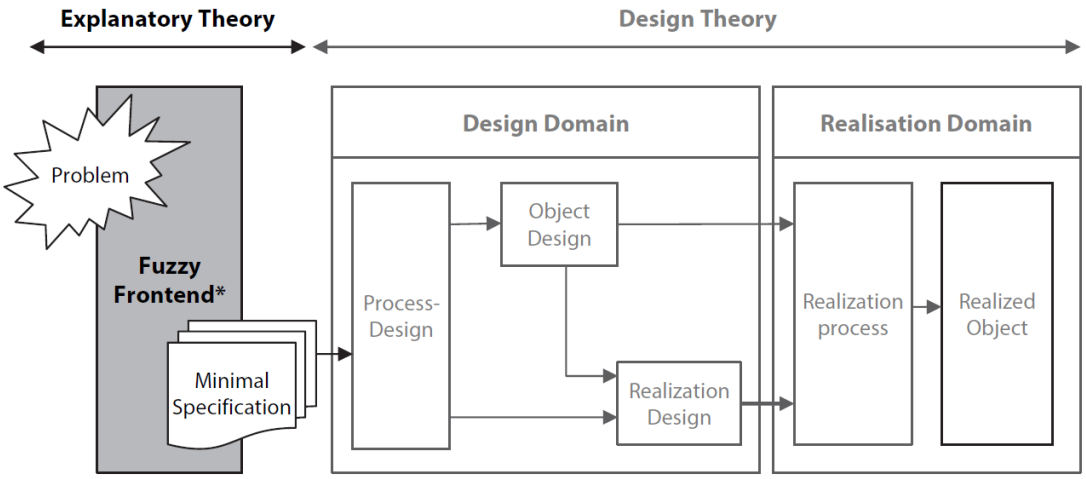


Figure 1.3: Ettema DSR for fuzzy problems, [72]

When searching for an explanatory theory, one starts by examining the interests, suspicions, and assumptions at the so-called fuzzy front end. This step ends in a first set of specifications which one uses in the first iteration of the design process [72].

1.5.4 DSR according to Wieringa

Design science, according to Wieringa, is the design and investigation of artefacts within a context that are designed to interact with this context and should improve something in that context.
"To do a design science project, you have to understand its major components, namely, its object of study and its two major activities. The object of study is an artefact in context..., and its two major activities are designing and investigating this artefact in context... For the design activity, it is important to know the social context of stakeholders and goals of the project, as this is the source of the research budget as well as the destination of useful research results. For the investigative activity, it is important to be familiar with the knowledge context of the project, as you will use this knowledge and also contribute to it. Jointly, the two major activities and the two contexts form a framework for design science." [24]
An artefact is an object that solves a problem by interaction with the context of that artefact. The same artefact may or may not solve another problem in another context by interaction with that context. Or in Wieringa's definition: *"An artefact is something created by people for some practical purpose. ... artefacts, are designed, and these designs are documented in a specification." [24]* Designing an artefact may result in multiple new artefacts that need to be designed. The resulting artefacts may answer knowledge questions. Thereafter the artefact must be implemented, validated, and evaluated. The implementation and hence the application of the artefact takes place in the original problem context, thereby validating that this application benefits the stakeholder goals if implemented. The final evaluation of the artefact, involves the evaluation of the results of the implementation with one or more stakeholders in the field. One can subsequently

use these results to improve the artefact.

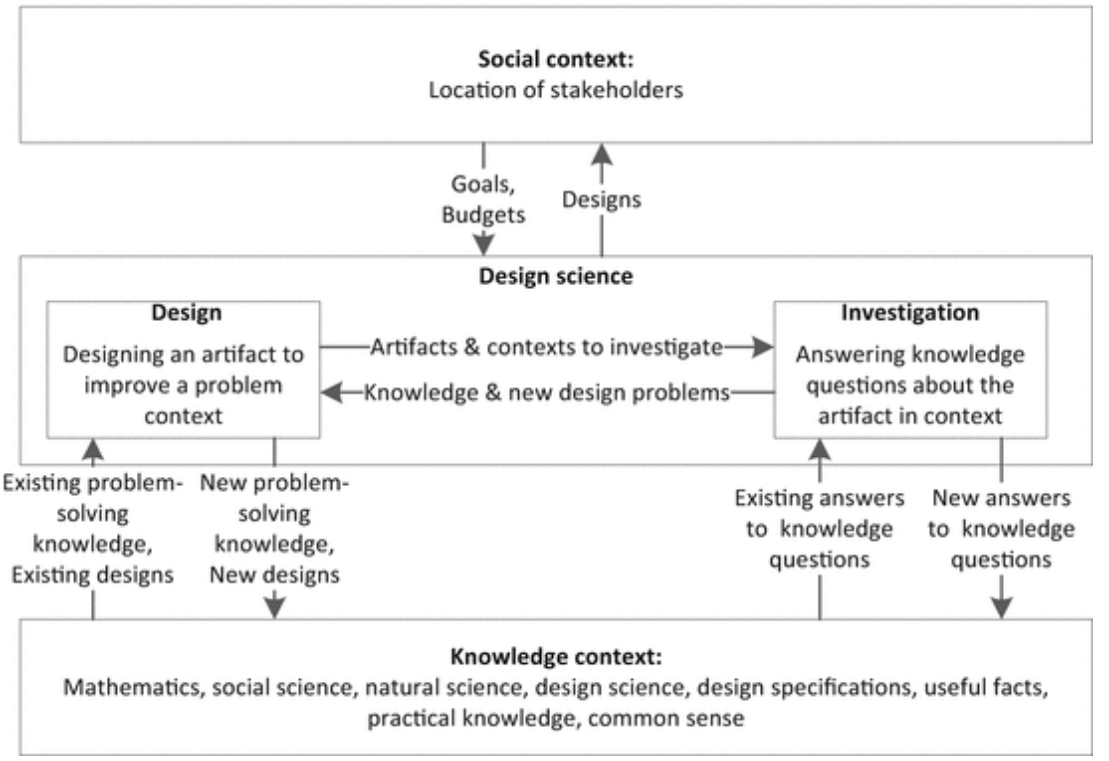


Figure 1.4: Wieringa DSR, framework [24]

1.5.4.1 Design and Engineering Cycle

While designing, one investigates, creates and validates the design of the artefact(s) in an iterative cycle. This problem-solving process cycle is a logical structure to be able to engineer and design an artefact. The main task in this cycle is to understand the problem and choose the right design to resolve the problem before implementation. Through evaluation, we can learn from this implementation for the next cycle [24].

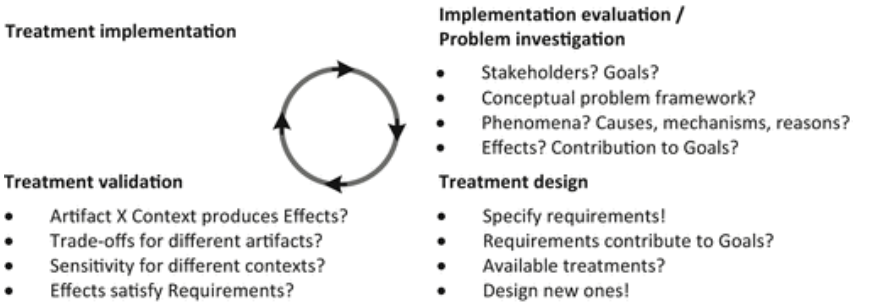


Figure 1.5: Wieringa DSR, engineering cycle [24]

Stakeholders A stakeholder consists of a person or group that is affected by the problem and the resulting artefact. The stakeholders' goal is to invest in the solution of the

problem. These goals originate from, possibly conflicting, desires and are the primary source for problem solution by engineers. They use the resulting artefacts in the original problem context. Using an artefact, may trigger new desires and consequently new engineering cycles [24].

Implementation Evaluation, and Problem Investigation Problem investigation retrieves the pre-involvement requirements, and the artefact design follows this process step. After the design implementation, the implementation evaluation investigates the achieved results of the use of the artefact. *"The goal of implementation evaluation and problem investigation is to build a scientific theory of real-world implementations and real-world problems, respectively."* The use of single-case experiments is useful in this context because they can provide deep insight into the mechanism of the behaviour of the artefact within its context [24].

Artefact Design Artefacts are delimited by stakeholders requirements who, in turn, may find it difficult to specify them. Therefore, the researcher has to formulate the requirements as part of the design. The researcher has to justify the choices made in the process by giving contribution arguments to each requirement [24].

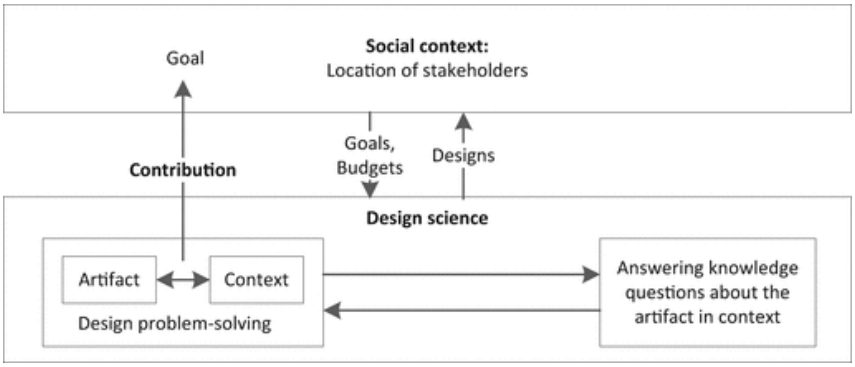


Figure 1.6: Wierings DSR, problem-solving [24]

One operationalizes the properties of the artefact by specifying tests or indicators for functional and non-functional requirements, respectively.

Artefact validation *"The goal of validation research is to develop a design theory of an artefact in the context that allows us to predict what would happen if the artefact were transferred to its intended problem context"* A validation model is a model between the artefact and the real world. This validation model is used to predict the usability of the artefact. The simplest validation model is an expert opinion by an expert panel. Additionally, the single case experiment will also provide validation information. In all cases, the researcher starts with laboratory conditions; scales come in at a later stage. Finally, research should scale towards realistic situations.

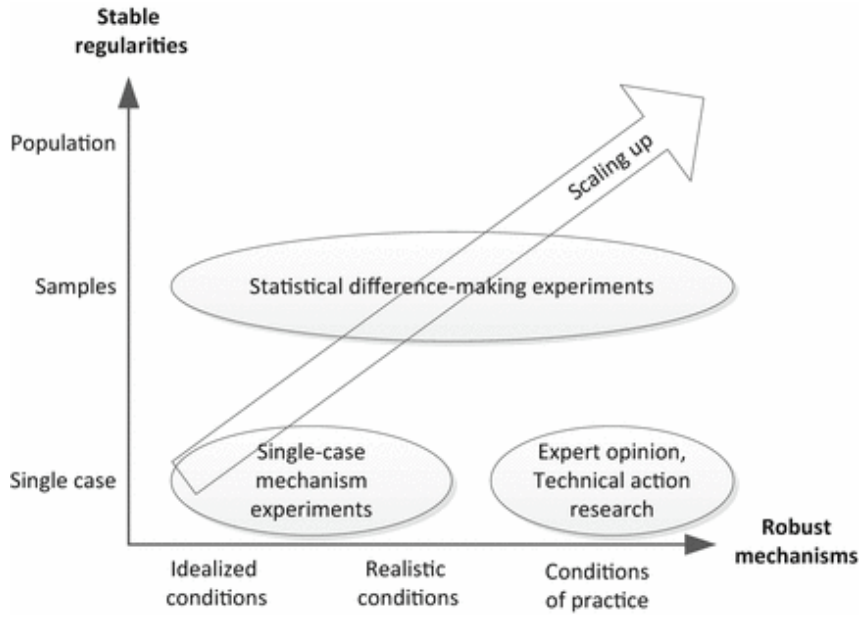


Figure 1.7: Wierings DSR, scale up [24]

Conceptual Framework Within the research context, one needs a conceptual framework to define structures. This framework, with definitions of concepts, is used to define artefacts and their behaviour, their properties and other phenomena. This framework can also be analysed in itself and is a "tools for the mind" [24]. Measuring the operationalized artefacts takes a lot of effort. Therefore, the artefact validity can be measured with respect to the research goal and research questions.

1.5.4.2 The Empirical Cycle

"The empirical cycle (fig. 1.8) is a rational way to answer scientific knowledge questions." [24] For this cycle, researchers need to know three things: goals, the desired improvements and the already existing knowledge base. They also consider the contribution of the artefact to the knowledge base and the improvement to practice.

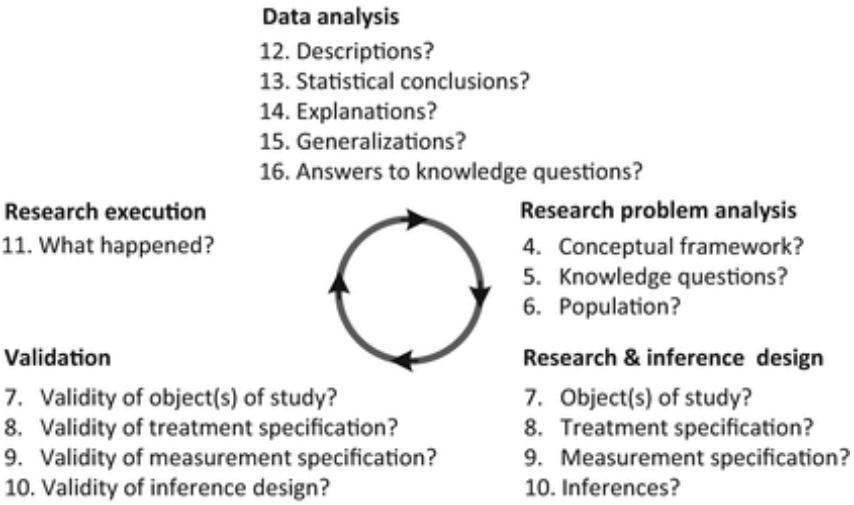


Figure 1.8: Wieringa's DSR, empirical circle [24]

The research context of a design science research project may be utility-driven or curiosity-driven. One tries to solve a knowledge problem and generalise it for an assumed population of interest. The empirical cycle helps to focus on this task but is no checklist. Two rules hold: created knowledge is only present after research execution and any knowledge present may influence the research. Therefore, all events that could have influenced the conclusions must be stated.

1.6 Thesis Structure

To find all relevant information on the subject of DEMO, the EE community functioned as a solid base. The 2014 Enterprise Engineering Working Conference (EEWC) gave insight into the information that is available within this community. The information in DEMOSL 3.7 [1], DEMOSL's origin Enterprise Ontology (EO) [25] and its successor The Essence of Organisation (TEoO) [26] contains ample references to early work on DEMO theory. Moreover, all twelve proceedings *'Advances in Enterprise Engineering'* contain relevant information about the attempts for progress and the history of DEMO. Searching in the knowledge base of the domain makes all findings semantically relevant.
Subsequently, we consulted references to connecting subjects such as meta modelling [75], set theory, logic, and modelling.
Some results from the DEMO knowledgebase are Wang's exchange model [76] and Figueira's action rules [77].
Initially, we considered building a new computer-aided design tool with the necessary modelling and verification support. To stick as closely as possible to the design of DEMO, we chose to adopt the DEMOSL documentation as the backbone of the automation and hoped to have all rules, legends, and the metamodel of DEMO. This first attempt at a tool based on the metamodel that can accurately represent DEMO models turned out to have omissions. These omissions triggered our research on the investigation of the full DEMOSL information.

Explained in the information in the previous sections 1.5.2 to 1.5.4 we concluded in section 1.5.1 that Wieringa's DSR approach [24] matches our research objectives most.
In this thesis we have the objective of improving DEMO's metamodel and its formal representation, the validation of metamodels and the tooling of the DEMO methodology. We defined these artefacts as the object of our study. The social context of the framework was the DEMO professionals and their organisations. They used the method and needed the results of the analysis. These artefacts and their context are visualised in fig. 1.6. The knowledge base referred to by DSR was mentioned in sections 1.2.2 and 1.6. In the design cycle, we chose to validate most artefacts, not to evaluate them. The only artefact that we evaluated is the exchange-metamodel (see section 4.1.4). The artefacts that this research provided are a DEMOSL proposal, containing an improved metamodel with mathematical rules, a verification model, and a Sparx Enterprise Architect (SEA) model that enables one to model DEMO in a tool. These models and rules are interconnected (see section 8.2.5) and have been tested (see Appendices).
We started this research on the observation of limited automated support of the DEMO modelling. We considered building new automation with the necessary modelling and validation support. To stick as closely as possible to the design of DEMO we chose to adopt the DEMOSL [1] documentation as the backbone of the automation and hoped to have all rules, legends, and the metamodel of DEMO. This first attempt at a metamodel that supposedly accurately represents DEMO models turned out to have omissions. This triggered the investigation of the DEMOSL information. After we had described all the omissions, we continued with the expansion of DEMOSL to support the specification of automated storage and exchange of DEMO. We wanted to find all these omissions to make the method better applicable and internally aligned. Therefore, we expanded the specification and described most model aspects, thereby making the DEMO specification more complete. This expansion included not only the ontological-metamodel and verification rules but also the visualisation-metamodel needed to exchange the representations of a DEMO model. This completed the problem investigation as described in section 1.5.4.1.
To be able to design the exchange model, we needed support from the community that would be using the exchange model. Therefore, during the creation of the exchange model artefact, a focus group commented on the usability of the artefact. This focus group consisted of experts in DEMO modelling and experts in the field of process modelling in VISI, which is a derivate of DEMO. The VISI experts had a software development background and focussed on the usability of the exchange model in their products. This group discussed and gave insight into the usage of the DEMO models and the requirements for the exchange metamodel (see fig. 1.6).
The next logical step was to find a tool because an exchange model has only value in an automated environment. Therefore, the specification of the metamodel and subsequent implementation of a DEMO modelling tool was the carrier of the research project.
Next, we verified this metamodel with existing DEMO models. Here, the usage of automation and the implementation in an organisation gave us feedback on modelling using the tool, and on the results of the diagrams we created.
Although the theoretical side of DEMO was very well-founded, to make the method more usable, we considered the practical usability of the method and metamodel essential, as a

whole and for the aspect models themselves. Although various case studies showed the use of DEMO [78, 79], the models were never integrally represented in the results nor were all aspect models used. Therefore, we conducted several case studies to improve the usability of the aspect models as well as the integral model and the repository (see section 1.5.4.1). The case studies concerned a real estate organisation, a health care organisation and some logistical organisations. During these intervening case studies, we tested all artefacts. We used it to store the DEMO model to find an optimal, practical implementation that can automatically verify the model. We have modelled these organisations and they served as case-study subjects to show how the complete metamodel of DEMO can help to model organisations and reuse DEMO models for their organisation.

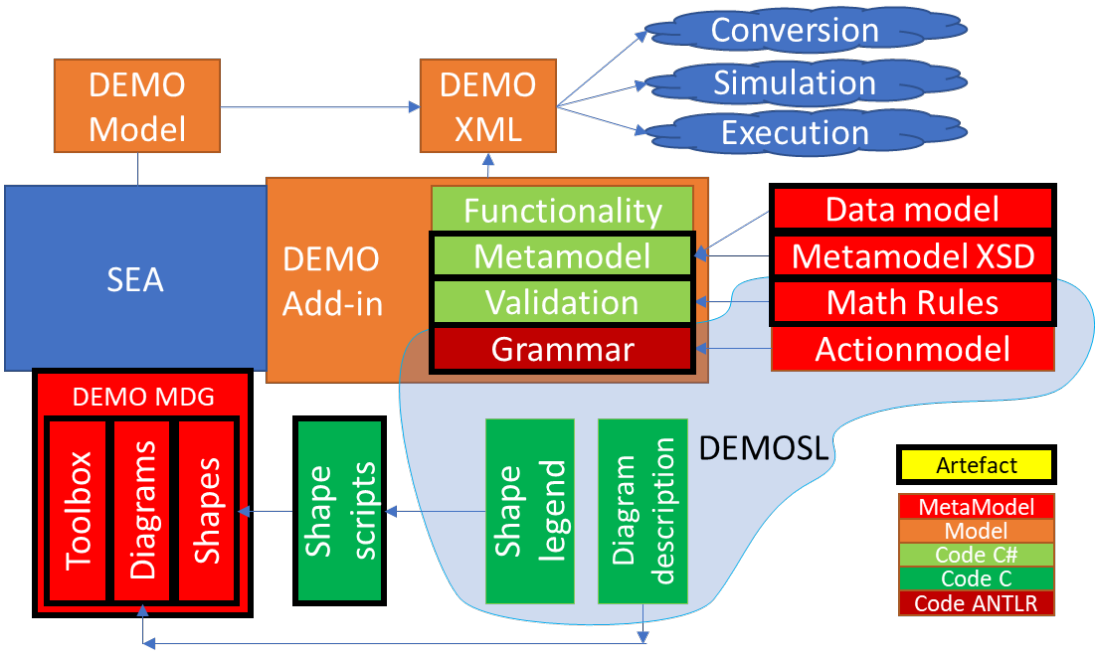


Figure 1.10: Thesis Artefacts in their Environment

We performed the case studies as parallel single-case mechanism experiments (see fig. 1.7) [24], mostly during implementations of Enterprise Resource Planning (ERP) systems within the selected organisations. These ERP systems tend to have the most substantial impact on organisations and have a significant failure rate. Improving the implementation of ERP can be a significant contribution of DEMO to the world of Information and Communication Technology (ICT).
These steps were iterated several times during the research project (see section 1.5.4.2)
Finally, we investigated the usability of this automation for building new DEMO models together with other technologies and the representation of these models.
This thesis contains the results of the research conducted to obtain the PhD title.
Section 1.5 completes the introduction to this thesis.
Part I of the thesis describes the conceptual model. We start by explaining the DEMO methodology and other notions in such detail that the artefacts can be understood. After

the introduction to the DEMO methodology, we evaluate the DEMOSL as-is, to find most omissions from an automation standpoint. Starting from chapter 4 the metamodel DEMOSL will be discussed in detail with all (logical) verification rules. The formalised language, syntax, and semantics will be discussed per aspect model, diagram, and element. Following the metamodel, section 5.3 shows all implemented business rules on the metamodel. Chapter 5, which is the last chapter of Part I, describes the tool choice and implementation of the metamodel in the add-ins on that tool. All artefacts in Part I have been summarised in fig. 1.10. This figure gives an overview of the artefacts and their dependencies.
Part II describes all case studies that were conducted to find or create DEMO models in the real world. Five cases (i.e. real estate management; retail sale; medical production firm) describe their particular modelling needs for their branch or organisational structure. Chapter 6 ends with some practical problems that currently cannot be described correctly in DEMO, but that we do need to register for future research.
Part III will revisit the research questions of section 1.4.2. After that, we will discuss the conclusions of the research and present recommendations for tool use, methodology adaptation, and practical usage of DEMO. Chapter 7, the last chapter of this thesis, contains suggestions for future research.
The appendix lists the full implemented metamodel for reference purposes, including some proprietary content. We did not add the appendices to the printed version of the book. Readers that wish to go through all the appendices are welcome to request the pdf version of these appendices.

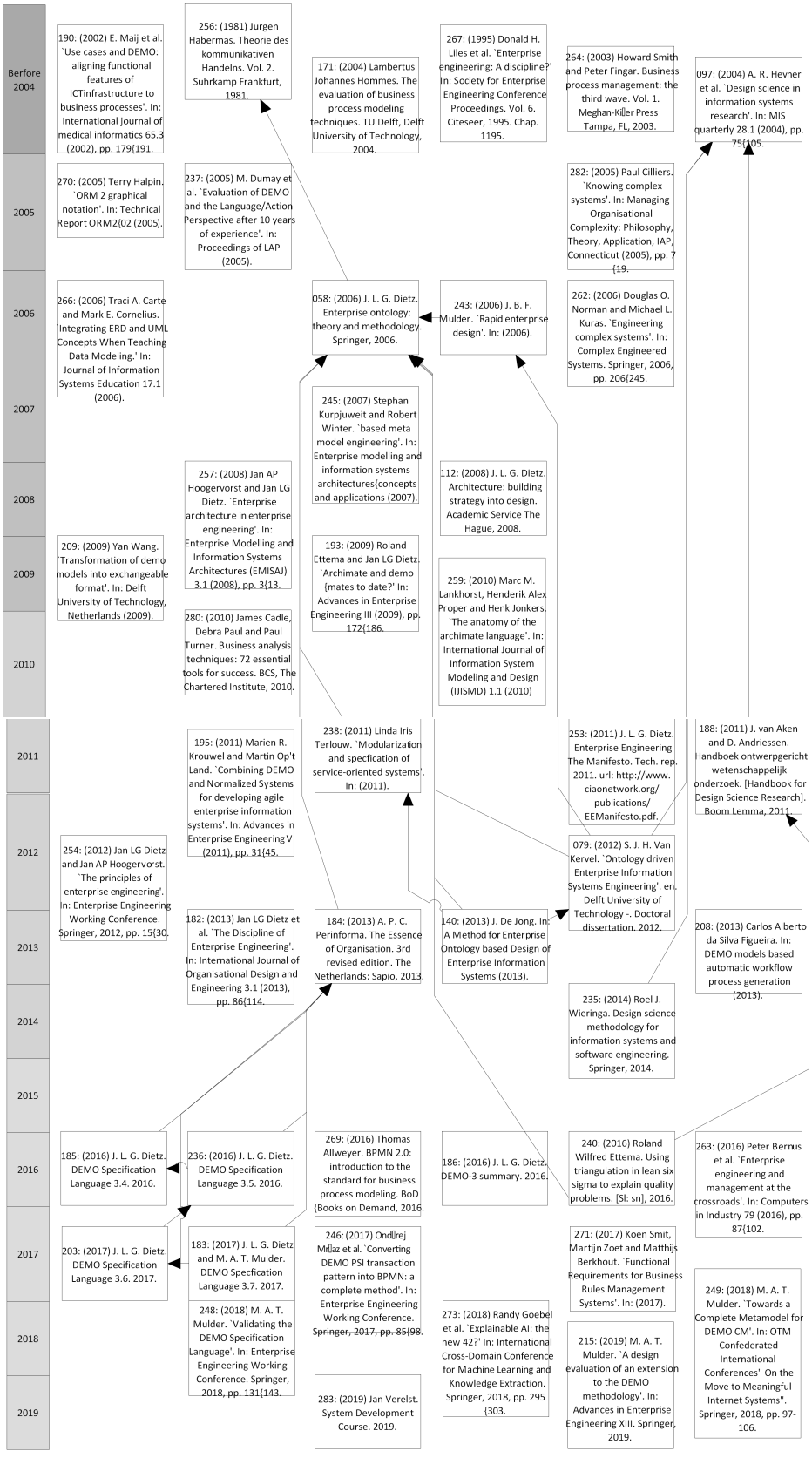


Figure 1.9: Literature and their dependencies

Part I

Conceptual Model

Chapter 2

Conceptual Framework

In this chapter, we will look into DEMO, the Design and Engineering Methodology for Organisations that is part of Enterprise Engineering (EE) as was described in section 1.2.2. We will also look into the definitions and notions that are required to understand the artefacts that have been created.

2.1 DEMO

As we explained in section 1.2.2, DEMO is a methodology with a rich history. DEMO 3, as used in this research, is based on the Performance in Social Interaction theory (PSI-theory). We will first summarize the most relevant parts of the PSI-theory to establish an understanding of the theory in the context of our research about the DEMO metamodel [25]. Although we acknowledge that the theory presented in these sections is very compact, our research does not focus on these parts of the theory and, therefore, we decided to focus on the DEMO Specification Language (DEMOSL) part. Moreover, this thesis is not a DEMO course and we will only briefly summarise the DEMO theorie that we think are needed to understand the rest of the thesis. One can study one of the books covering the theory of DEMO [25, 26, 9] to understand the whole methodology.

2.1.1 PSI-Theory

The PSI-theory constitute the foundation of Enterprise Ontology (EO) from a construction perspective. The PSI-theory divides an organisation into acts and facts that reach commitments and agreements in predictable phases. One composes these agreements in a logical order that demands the right responsibilities of the right subjects. Finally, the subjects' capabilities are appointed into three categories to make distinctions in decisions and other actions. This standard pattern and production can be used anywhere within the organisation to enhance communication between subjects. We call this pattern a transaction.
The PSI-theory divides the world into two worlds: communication (c-world) and products (p-world). Each of the c- and p-worlds may perform acts and produce facts. The communication acts, performed by actor-roles, produce communication facts. The same rule

holds for the p-world. In short, actors perform communication and production acts, thereby creating communication facts about the creation of production facts.
Furthermore, the PSI-theory defines the communication pattern between two actor roles. Every commitment to reach a production result follows this universal pattern (see fig. 2.1). A transaction evolves in three phases: the order phase with four communication steps, the execution phase with one production step and the result phase with four communication steps. Whenever the request, promise, state or acceptation relating to the product changes, one can revoke those steps to be able to change the commitment. The product (p-fact) becomes existent once the actor roles have performed at least all steps of the happy flow of the transaction as drawn in the middle section of fig. 2.1.
From the context of an organisation, an external actor role starts a transaction or self-started actor role inside the organisation. Furthermore, every transaction can start another transaction as part of its process flow, thereby creating a tree of executed transactions.
The transactions are divided into so-called performa, informa, and forma levels. These levels describe the typical activities of subjects to decide, calculate, and store, respectively. The transaction ontology can be used for a service-oriented system organisation [80]. The specification alone though is not enough to implement the services in a usable form. At the system boundaries, it must be possible for humans to interact with the system where these interactions must follow the universal transaction pattern as well as to communicate within this pattern in a human-comprehensive manner. Therefore, one realises the interface design around the transaction; also, all system transactions are designed in a similar way and with the same universal transaction pattern in mind. This specification allows one to communicate with the protocol but does not force the communication to be the same on both sides of the transaction.

2.1.1.1 Universal Transaction Pattern

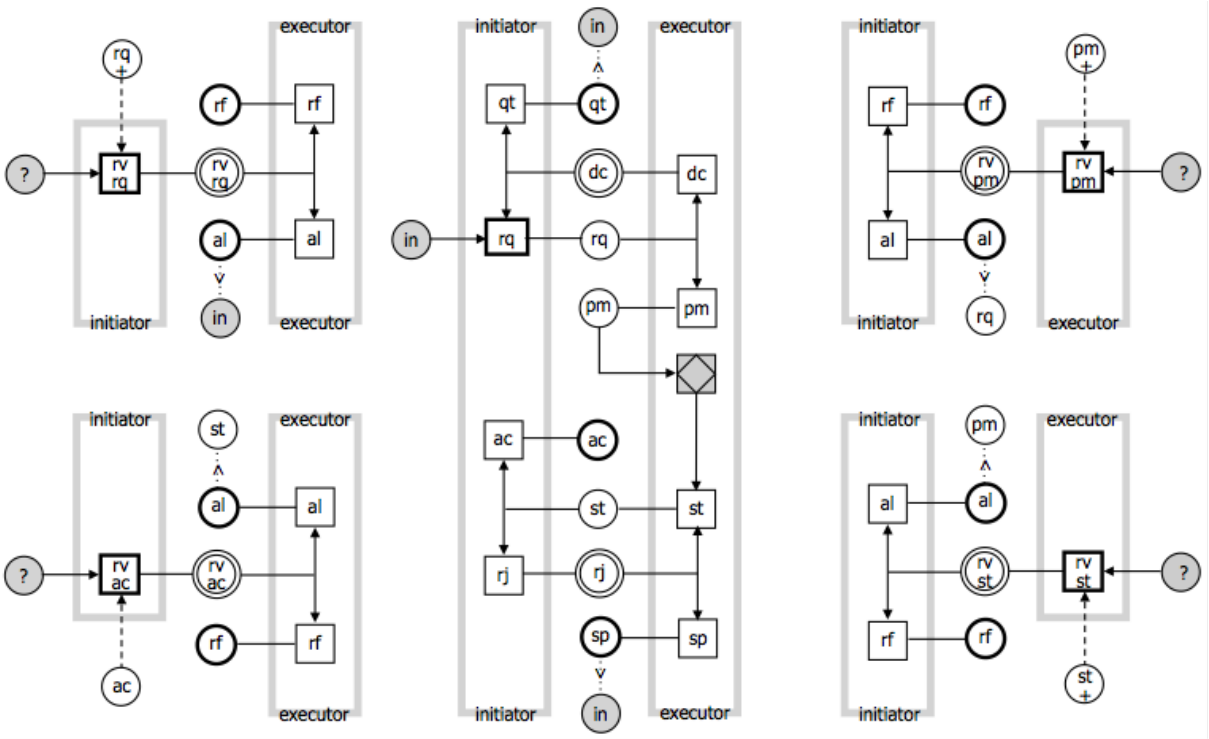
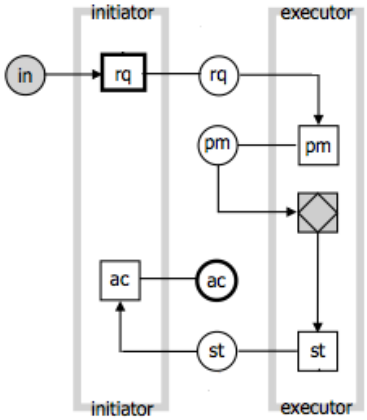


Figure 2.1: DEMO 3 universal transaction pattern

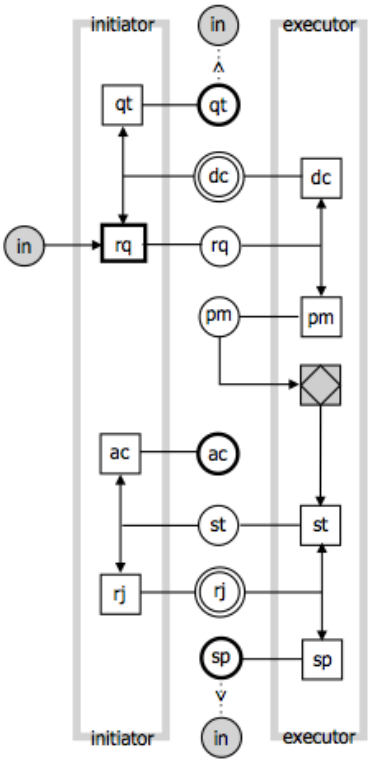
In the base of a transaction, the initiator starts the communication to obtain a product, and the executor communicates and produces to deliver this product. We will now in detail go through this universal transaction pattern at three levels: **Happy flow**, **Standard flow**, and **Revocation flow** First, we will examine the happy flow, which consists of four communication steps and one execution step. These steps are in bold in the diagram. Next, we will enhance our view of the discussion paths and add four more potential communication steps, the revocation step. Finally, we will include the three revocation steps of each of the four possible types of revocation.



Legend: initial(in), request(rq), promise(pm), state(st), accept(ac)

Figure 2.2: Happy flow

Happy flow The happy flow as illustrated in fig. 2.2 consists of four communication steps and an execution step. We will explain these steps in three phases: negotiation, execution, and result phase. The first phase is the negotiation or order phase. The next phase is the execution phase, followed by the final phase, the result phase. In the order phase, the initiator sends a request to the executor. Upon receiving the request, both initiator and executor negotiate the requirements for execution. The executor has predetermined the form of the requirement (as in ordering from a menu), and the actor roles only negotiate whether or not the request is within the boundaries of the menu. After the negotiation on the requirements for execution, the executor approves the request. In the happy flow, we assume that the executor can fulfil the request, and can promise that the request will be executed and when an answer will be given. In the execution phase, the executor does the actual processing of the request that is needed to provide a result. After the negotiation and execution, the final phase is due, the result phase, and the executor states the result, after which the initiator accepts the result. Hereafter, the communication cycle is complete, the initiator has received the requested product, and the executor probably waits for the next initiator to start this process all over again.



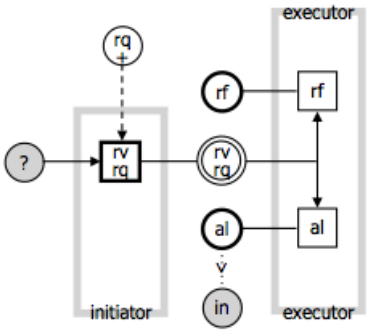
Legend: initial(in), request(rq), promise(pm), state(st), accept(ac), decline(dc), quit(qt, reject(rj, stop(sp)

Figure 2.3: Standard flow

Standard flow In the standard flow as illustrated in fig. 2.3, we can identify four steps in the order phase, we have the same execution phase and four steps in the result phase. In this standard flow every actor role operates in turn. Because this is normal behaviour we have called this flow standard. First, we take a look at the order phase. In the order phase, the initiator makes a request for a product. When the executor can grant this request (e.g. he/she believes he/she can execute), he will promise the requested product, but when the executor cannot promise, he has to decline the request. The declined state is a discussion state where the initiator can either quit and consider the transaction as unsuccessful or alter the original request and resent it. These four steps are all part of the order phase, and the involved actor roles, initiator and executor, reach an agreement before the execution phase will start. After the conclusion of the order phase, the executor can execute and state the result after which the initiator can judge this stated[1] result. The initiator can either accept the stated result or reject the result which, again, is a discussion state between initiator and executor. In this last discussion state, the executor can decide to restate the (unchanged) result or stop the transaction unsuccessfully if the initiator does not accept the (re)stated result or if the executor is not able to convince the initiator of the (re)stated result.

[1] 'stated' has been replaced with 'declared' in DEMO 4

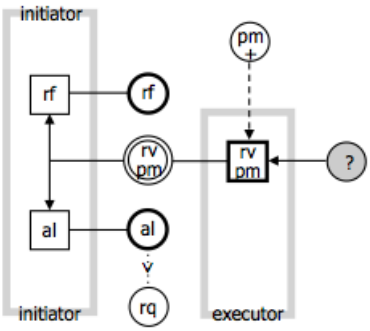
Revocation flow Until now, we have described both the happy flow in detail where everything goes according to plan and the standard flow where we described the order and result phase in more detail. Nevertheless, what happens if one wants to cancel one's request when the executor is still in the execution phase? That is the situation where the four revocation patterns are applicable. The patterns provide a way to revoke actions in the communication, no matter in what phase the communication cycle is. The response to this revoke is what makes the pattern interesting. The executor of each revoke can either refuse or allow the revoke. In the worst case scenario, all revocations can be active at the same time, making the pattern very complex to understand.



Legend: revoke request(rvrq), refuse(rf), allow(al), initial(in)

Figure 2.4: Revoke request

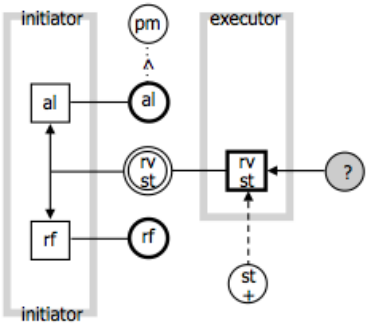
Revoke Request The first revocation flow as illustrated in fig. 2.4 is the revoke of a request by the initiator. This describes the withdrawal of the request, and when this withdrawal is allowed, the communication path can only end in a quit. The result of this revoke is, as seen from the executor, as if the request never took place. After the revoke has been allowed, the initiator can continue with other business, and the executor has to end all sub-transactions that were part of the underlying transaction. The cascading consequences will not be discussed in this section.



Legend: revoke promise(rvpm), refuse(rf), allow(al), initial(in)

Figure 2.5: Revoke Promise

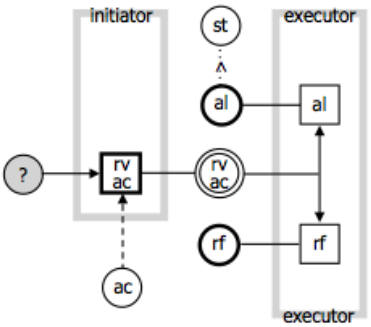
Revoke Promise Next to the initiator, the executor can also use revocation patterns on the execution side of the communication pattern. The revoke promise pattern as illustrated in fig. 2.5 allows the executor to revoke a previously promised request and proceed to the decline of the request. This situation applies when in the execution-phase, something unexpected happens that makes it impossible or undesirable to state the result.



Legend: revoke state(rvst), refuse(rf), allow(al), initial(in)

Figure 2.6: Revoke State

Revoke State The last revocation pattern is the revocation of a statement in fig. 2.6 where the result, once stated, is no longer valid. If this revoke is allowed, the promise must be made again for a changed answer. This sequence is a revocation pattern that goes from the result phase back to the order phase. In this situation, the actor roles will not redo the negotiations but will repeat the promise, so that the executor can redo the execution (e.g. to promise a new delivery time).



Legend: revoke accept(rvac), refuse(rf), allow(al), initial(in)

Figure 2.7: Revoke Accept

Revoke Accept Revoke an acceptance as illustrated in fig. 2.7 means that the initiator is returning to a previous accept step. The initiator has already accepted the result but, on second thoughts, he is not satisfied with the result and revokes the previous accepted step.
This concludes the list of steps of the universal transaction pattern for DEMO 3. We have left out the complete explanation of all steps and when and why you can use them. That exercise is beyond the scope of this thesis and the reader is encouraged to do that exercise in either [25] or in [9].

2.1.2 DEMO Concepts

DEMO has just a few basic concepts: Transaction Kinds (TKs) and Elementary Actor Roles (EARs). We will refer to these concepts mostly by their name or abbreviation.
The TK is the abstraction of the universal transaction pattern and represents the communication and production acts and facts. The EAR is the representation of the actor roles in the a-world. Two aggregated versions of these two concepts are present in the form of the Aggregate Transaction Kind (ATK) and the Composite Actor Role (CAR). In both aggregated concepts, the underlying singular versions of the concept can be hidden. This enclosure within the ATK and CAR is a viewpoint on an organisation part that does not have to be revealed yet. This hiding of the organisation can be described from the viewpoint of the c/p-world.
The earlier described steps of the universal transaction pattern map also to a concept of DEMO. In this thesis, we will call them Transaction Process Step Kind (TPSK). These are smaller process steps than a transaction in the c/p-world that we will be able to model.
In the p-world, there are more concepts that need to be addressed. The Entity Type (ET) with its Attribute Type (AT) are concepts that we use to model the information needs in an organisation. The construction of these types fulfils most needs in information modelling, and we will limit the DEMO overview to these concepts. A completer list of DEMO concepts can be found in chapter 4.

2.1.3 DEMO Aspect Models

The Design and Engineering Methodology for Organisations (DEMO) [25] methodology can produce a so-called essential model of an organisation that forms the integrated whole of four aspect models: the Construction Model (CM), the Action Model (AM), the Process Model (PM), and the Fact Model (FM). A combination of diagrams and cross-model tables can express the whole DEMO model.

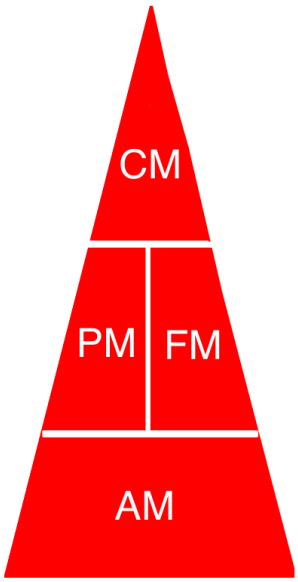


Figure 2.8: DEMO aspect models

The CM is the first and the most comprehensive model to produce when modelling an organisation in DEMO, applying the Organisational Essence Revealing (OER) method. This method is explained and enhanced in section 6.7. A CM is a model that represents the construction of an organisation (or better: of a Scope of Interest). This model consists of the identified transaction kinds and the actor roles that are either executor or initiator of these transaction kinds. The resulting 'network' of transaction kinds and actor roles is always a set of tree structures, which arise from the inherent property that every transaction kind has exactly one elementary actor role as its executor (and vice versa), and that every actor role may be the initiator of none, one or more transaction kinds.
A CM can be expressed in an Organisation Construction Diagram (OCD), a Transaction Product Table (TPT), and a Bank Contents Table (BCT).
The OCD is a graphical representation of the identified TK, ATK, EAR, CAR within a Scope of Interest (SoI), and the links between them which show the dependencies between roles in execution and information. These links have three appearances, the initiator and executor links, and actor roles may also connect to transaction kinds through information links. The latter express that the specific actor role has (reading) access to the history of all transactions of the transaction kind with which it connects. Therefore, one can also interpret the transaction kind shape as a transaction bank. The high abstraction level makes this OCD a compact diagram in relation to the implementation of the organisation.
The TPT shows the TK identification and description together with the product identification and description. This table is used to get insight into the products that are being created in the organisation.
Finally, the BCT shows the contents of the ATK. This contains the identification and name of the ATK and the ET and attribute types of those ETs that are present. This is used to show the extend of (external) data.

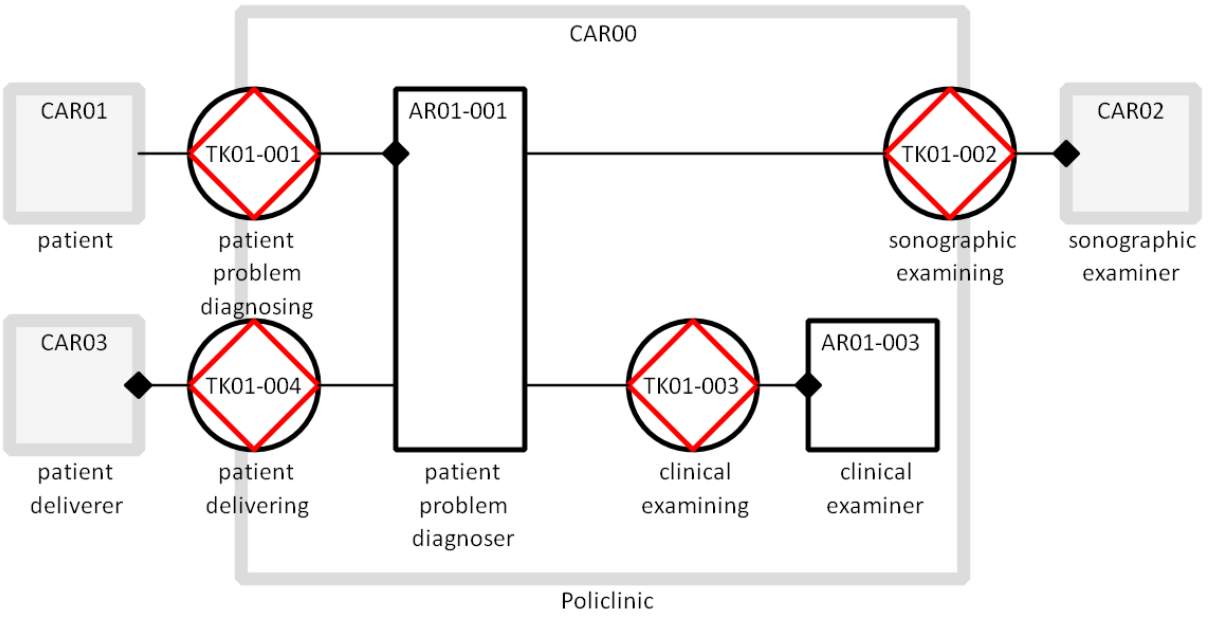


Figure 2.9: Poligyn OCD modelled in the tool

Name	Alias	Product Kind Formulation
TK01-004	patient delivering	the patient of [patient problem] is delivered
TK01-003	clinical examining	[clinical examination] is performed
TK01-002	sonographic examining	[sonographic examination] is completed
TK01-001	patient problem diagnosing	[patient problem] is diagnosed

Figure 2.10: Poligyn TPT modelled in the tool

We modelled the example case of [9, chapter 17], named Poligyn using the tool. In fig. 2.9 we see the four TK of the CM with its initiating and executing actor roles within the modelled organisation. These TKs can also be shown in a table as the TPT in fig. 2.10. One step further in the analysis, we get to the process dependencies depicted in fig. 2.11. Finally, in fig. 2.12 the FM shows the internal references between ETs and the relations between ET and TK.

The PM of a Scope of Interest bridges its CM and the coordination part of its AM. To this end, it specifies how the transaction kinds in a tree are related to each other. More precisely, it specifies which transaction steps in an enclosed transaction kind are connected to which steps in the enclosing transaction kind, and specifies by

which kind of link (response-link or wait-link) they are connected. A PM is expressed in a Process Structure Diagram (PSD) which shows the relations between the process steps of interrelated transactions, and (optionally) in one or more Transaction Pattern Diagrams (TPDs). These diagrams are ways to visualise the dependency of transactions and transaction steps, respectively. Business rules are partially covered as well.

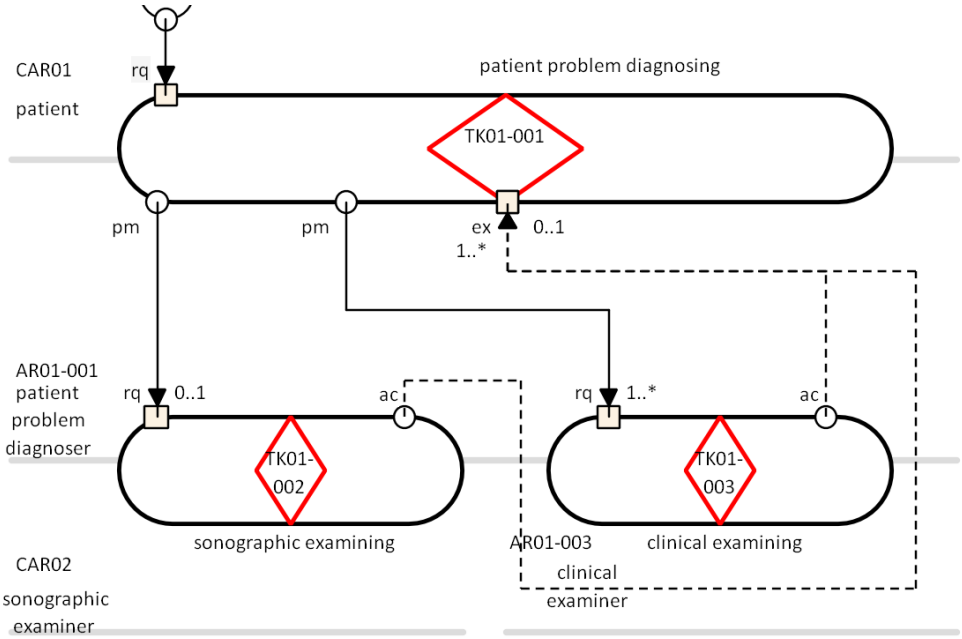


Figure 2.11: Poligyn PSD modelled in the tool

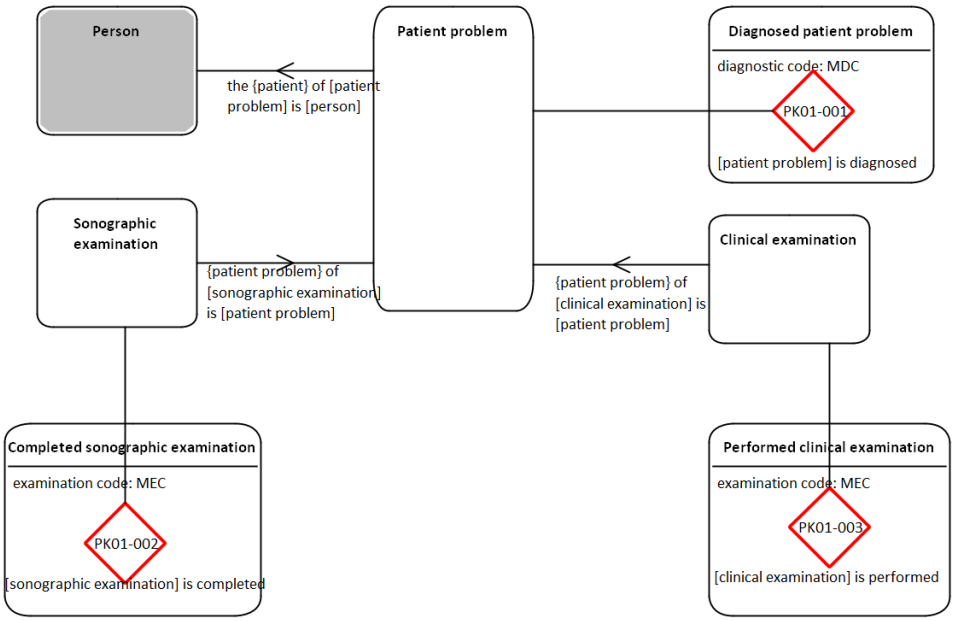
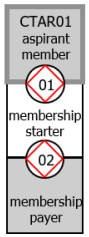


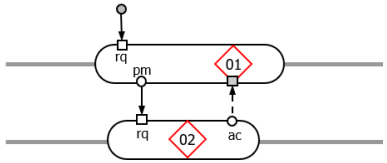
Figure 2.12: Poligyn OFD modelled in the tool

The FM of a Scope of Interest bridges its CM and the production part of its AM. To this end, it specifies the various entity types, property types, and attribute types, as well as their mutual relationships. The FM involves a single diagram kind, the Object Fact Diagram (OFD), which shows ETs and Product Kinds (PKs), and the Information Use Table (IUT). This model is often called the data model, although it shows much more information.

The AM of a Scope of Interest comprises the guidelines that guide actors in doing their work, i.e. performing their coordination acts and their production acts. Action rules, which are actually (imperative) business rules, guide actors in responding to the coordination events they have to deal with. A semi-structured English-like language is used to express the action rules. Work instructions guide actors in performing the production acts, i.e. in bringing about the products of transactions.
This last aspect model contains Action Rules Specification (ARS) and Work Instruction Specification (WIS). Per non-trivial process step minimal one specification shows the input and conditions to proceed in the transaction pattern or advance to other transactions. This specification is used to model all details of the (to-be) organisation.
Different aspect models contain overlapping elements; therefore, the DEMO *essential model* is the result of the combination of all aspect models.
The studied version of DEMO is called DEMO 3. It is published in [26] and in [1]. Examples in the context of this section are illustrated for diagrams in fig. 2.13 on page 64 and for tables in fig. 2.14 on page 65.

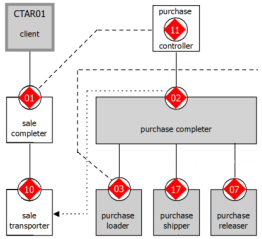
What is it: Diagrams



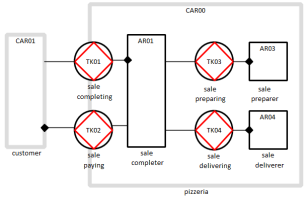


PSD

CSD

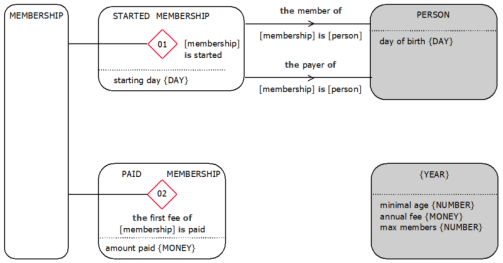


OCD

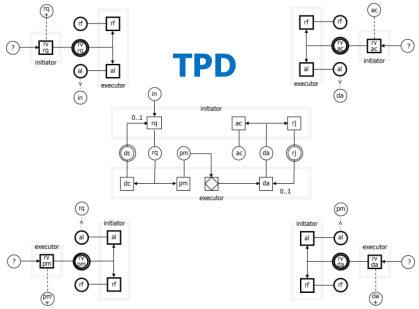


The Enterprise Engineering Company 2

OFD



TPD



CSD Coordination Structure Diagram

OCD Organisation Construction Diagram

Process Structure Diagram

Object Fact Diagram

PSD

OFD

TPD Transaction Pattern Diagram

DFS Derived Fact Specifications

ARS Action Rule Specifications

WIS Work Instruction Specifications

DFS

the duration **of** [rental] = **the** ending day **of** [rental] **minus the** starting day **of** [rental]

the rental charge **of** [rental] = **the** base charge **of** [rental] **plus the** fine charge **of** [rental] **minus the** accepted deposit amount **of** [rental]

the base charge **of** [rental] = **the** duration **of** [rental] **times the** daily rate **of the** car group **of** [rental] **in the** year **of the** starting day **of** [rental]

ARS

when rental completing **for** [rental] is promised
while deposit paying **for** [rental] is accepted

assess
rightness: * no specific condition *
sincerity: * no specific condition *
truth: * no specific condition *

if *performing the action after* ***then*** *is considered justifiable*
then * select car for rental *
request car taking **for** [rental]

WIS

Example of Job Instruction Sheet (標準作業指導書の例)

Job Instruction Sheet	Part # 26-0012	Required Quantity: 550	Date: April 26, 2007	Dept./Location:	Team Leader:	Supervisor:
	Part Name Base Unit Assembly			Prepared By:		

#	Step	Quality Check: Sampling	Quality Check: Tool	Note	Time	Takt Time 42	Cycle Time 40	STD WIP 3	◇ Quality ◆ Safety ● STD WIP
1	*Get a work piece and set into fixture*			*With both left and right hand*	1				
2	*Remove finished & set new one*				2				
3	*Check appearance*	1/1	Mini Gauge		12				
4	*Remove finished & set new one*			*Clean up head for every cycle*	14				
5	*Remove finished & set new one*			*Ensure direction*	3				
6	*Check appearance & put into container*	1/1	–	*Check both sides*	8				
				Total	40				

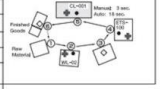


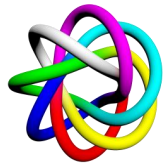
Figure 2.13: DEMO 3 and 4 diagram visualisations

What is it: Tables

transaction kind	product kind	executor role
TK01 rental completing	PK01 [rental] **is** completed	AR01 rental completer
TK02 car taking	PK02 **the** car **of** [rental] **is** taken	AR02 car taker
TK03 car returning	PK03 **the** car **of** [rental] **is** returned	AR03 car returner
TK04 deposit paying	PK04 **the** deposit **of** [rental] **is** paid	AR04 deposit payer
TK05 invoice paying	PK05 **the** invoice **of** [rental] **is** paid	AR05 invoice payer

TPT

The Enterprise Engineering Company 2



BAT

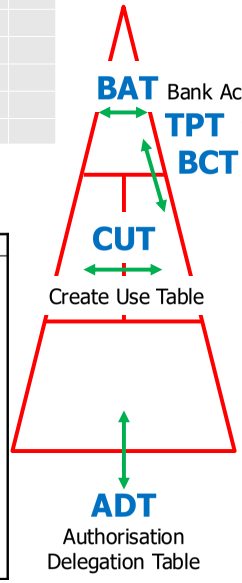
Bank Actor	TK01	TK02	TK03	TK04	TK05	TK06	TK07	TK08	TK09	MTK01	MTK02	MTK03
AR01	Ex	In								U	U	U
AR02		Ex									U	
AR03	U	In	In, Ex	U						U	U	
AR04	U			Ex				U	U	U	U	
AR05	U			In	In, Ex					U	U	
AR06	U			U		Ex	In	In	In	U	U	U
AR07							Ex				U	
AR08								Ex			U	
AR09									Ex		U	
CTAR01	In			In							U	
CTAR02						In					U	

BCT

bank	independent/dependent facts
TK01 membership starting	MEMBERSHIP [membership] **is** started **the** starting day **of** [membership] **the** member **of** [membership] **the** amount to pay **of** [membership]
TK02 membership paying	**the** first fee **of** [membership] **is** paid **the** amount paid **of** [membership]
MTK01 persons facts	PERSON **the** day of birth **of** [person]
MTK02 Volley facts	YEAR **the** minimal age **in** [year] **the** annual fee **in** [year] **the** max members **in** [year]

CUT

P-fact type	created in performing	used when settling
MEMBERSHIP	TK01/rq	
PAID MEMBERSHIP	<derived>	
PERSON	<given externally>	
YEAR	<given externally>	
[membership] **is** started	TK01/ac	
the first fee **of** [membership] **is** paid	TK02/ac	TK01/pm
the member **of** [membership]	<provided as parameter>	TK01/rq, TK01/pm
the payer **of** [membership]	<provided as parameter>	TK01/rq, TK01/pm, TK01/da
the starting day **of** [membership]	<provided as parameter>	TK01/rq
the day of birth **of** [person]	<given externally>	TK01/rq
the minimal age **in** [year]	<given externally>	TK01/rq
the max members **in** [year]	<given externally>	TK01/rq
the annual fee **in** [year]	<given externally>	TK01/rq
the amount to pay **of** [membership]	TK02/rq	
the amount paid **of** [membership]	TK02/da	TK02/da
the first fee **of** [membership]	<derived>	TK01/rq, TK02/da
the number **of** members **on** [day]	<derived>	TK01/rq
the age **of** [person] **on** [day]	<derived>	TK01/rq



ADT

P\T	TK01/da	TK02/ac	TK03/ac	TK04/ac
customer				D
order taker	A	A	A	A
baker				
student	D	D	D	

Figure 2.14: DEMO 3 and 4 table visualisations

2.2 Perspectives on DEMO Models

We need to distinguish between three perspectives on the models used when applying DEMO. These are illustrated in fig. 2.15.
The first perspective is the *methodology perspective*. Even though it provides a thorough theoretical basis, the DEMO methodology book [9] was primarily written to teach learners (students and practitioners) to create models in accordance with the DEMO way of thinking, and draw (human to human) communicable models to reason about the organisation. As such, the book puts the priority on "doing", when introducing the different model kinds. There was no need for a strict metamodel. Furthermore, the formalisation(s) provided in the book aim to support didactic goals rather than the development of automated modelling tools. As such, it was never meant to provide a detailed formalisation and metamodels, that would enable the development of, and automated support for, the methodology. As discussed in [81], different metamodelling and formalisation goals will and should also result in formalisations with different level(s) of detail and specificity.
When using DEMOSL as base for tool development, then it should indeed provide a more complete and detailed formalisation and metamodel; taking us to the *language perspective*. This is the perspective where it should be possible to (automatically) reason about models and metamodels, to e.g. support model verification. This requires the formalisation of the language definition in a format that lends itself better for automatic processing [81]. It also requires the completion and operationalisation of the ontological metamodels as included in the methodology perspective; in particular the rules and constraints to be applied to the actual models.
The last *model exchange perspective* concerns the exchange and storage of model information. Jointly reasoning about/with models is only possible when all parties involved have (basically) the same understanding of the model, which requires a joint understanding of the used notation.
The notation used must also be the same across different modelling tools. Therefore, the meaning of the elements and attributes must be well defined. Moreover, the notation must enable the reasoning within the exchange model to be able to check consistency of this model. In addition, the internal consistency needs to be correct and needs to be checked before the model can be converted back to the language perspective.
While a good theory should help professionals to do their job, more professionals are educated using automation as their main tooling. Where pencils were used, tablets have taken their place, and people see automation as their starting point. Tooling fulfils a lever function in broadening the use of the methodology. As a result, to enable tool development, a more complete and detailed formalisation and metamodel were needed.

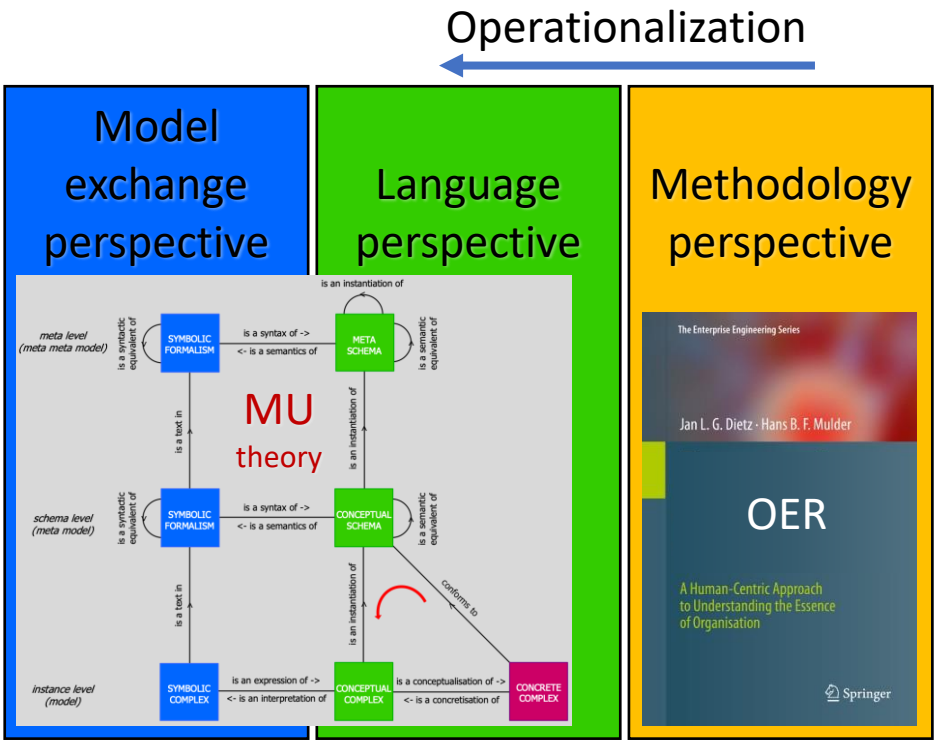


Figure 2.15: Perspectives on DEMO Modelling

2.3 MU Theory

The Model Universe (MU)-theory provides the theoretical foundation of the notions of model, modelling, and modelling language as used in DEMO. The DEMO methodology was published with its metamodels in various versions [25, 26, 1, 27, 82, 83] which describe how to contain the models for the CM, PM, FM, and AM. The metamodels, referred to as DEMOSL, describe the meta-level of the respective models but are not entirely aligned. The most recent version of the MU-theory has been presented in [9].
Given the aim of this thesis to evolve the accompanying DEMOSL, the MU-theory is considered as a fixed and pre-defined part of the knowledge base (in design science terms). In this section, we highlight some of the key elements of the MU-theory that are relevant to the understanding and positioning of the thesis.
The MU-theory theory adopts Apostel's [84] definition of model: "*Any subject using a system A to obtain knowledge of a system B, is using A as a model of B.*" This definition conveys the basic understanding of the notion of model as being a role [9].
A key part of the MU-theory is the General Conceptual Modelling Framework (GCMF), as depicted in fig. 2.16. The MU-theory uses the term complex to refer to systems in the general sense, as used in Apostel's [84] definition of model. However, since DEMO uses its own (more specific) definition of system, the term *complex* is preferred when speaking about modelling in general (as is the case for the MU-theory).

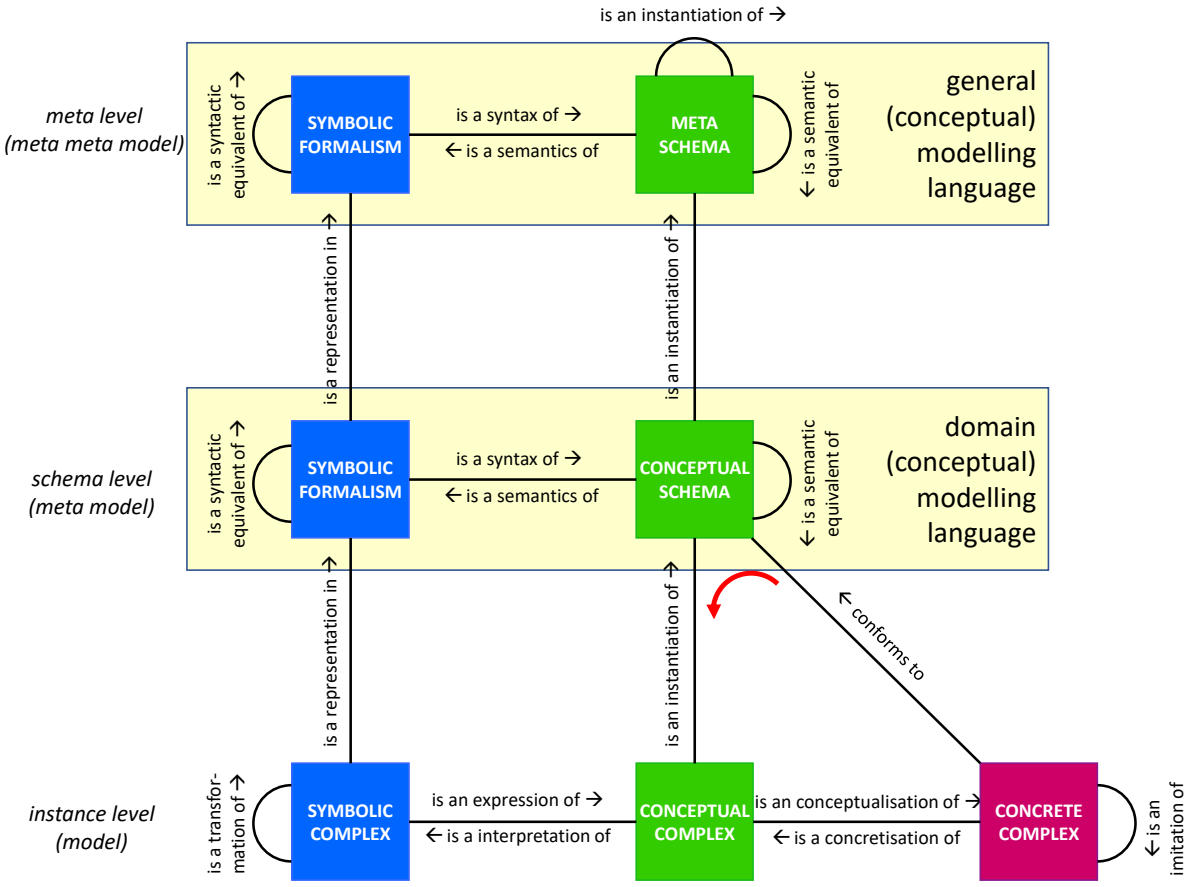


Figure 2.16: The General Conceptual Modelling Framework, adopted from [9]

The basic thinking underlying the GCMF is based on the semiotic triangle by Ogden and Richards [28]. In line with this, the MU-theory refers to the phenomena we observe and interact with in reality as *concrete complexes*, while the conception of these complexes an actor harbours in their mind is referred to as the *conceptual complexes*, and a resulting symbolic representation (such as a diagram, a narrative description, or an XML document) as *symbolic complexes*.
When combining this with Apostel's definition of a model, then one can state that when such a complex A is used to obtain knowledge of a complex B, then A is said to be a model of B. As a corollary to this, one can (using the terminology from the MU-theory) distinguish between *concrete models*, *conceptual models* and *symbolic models*.
The MU-theory states that the creation of a conceptual complex needs to conform to a *conceptual schema*. It is furthermore assumed that a *conceptual complex* can only be created in the mind of some actor observing a domain, if they already have a *conceptual schema* in terms of which they can observe the world. This is indicated by the red curved arrow in fig. 2.16. As such, a *conceptual schema* limits the observable world (of an actor).
The MU-theory also states that a modelling language essentially involves a combination of a conceptual schema and a corresponding symbolic formalism that provides the symbolic representation of the conceptual schema.

The DEMOSL is targeted at the schema level of fig. 2.16. In other words, it should provide a symbolic formalism corresponding to the conceptual schema(s) used by the different model (and diagram) kinds within DEMO. As illustrated in fig. 2.16, the MU-theory also identifies a meta level involving a meta schema and a corresponding symbolic formalism. This is where, in our case, we will find UML class diagrams and XML Document Type Definitions (DTDs) to define the DEMOSL.
The MU-theory also provides global guidelines on how to create the various levels of complexes and schemas. Finding a lower level from a higher level can be accomplished by deduction, which also provides a good method for the verification of models. Induction can be used to derive the meta level from examples. In doing the latter, it can be helpful to transform graphical languages into structured textual languages, called verbalisation. We used both deduction and induction methods to create metamodels and DEMO information models. We are using the conceptual schema (i.e model) of the organisation and we will express it in symbolic formalism (e.g. diagrams, tables). The conceptual model will be formulated using a meta schema (i.e. metamodel).

2.4 Repository

A repository, in the context of modelling, is a storage where all symbolic complexes of the defined metamodel can be stored. To be able to create a single repository that stores the whole model, it is indispensable that the aspect models can be merged uniformly without compromising the methodology. Therefore, in this thesis, the separate aspect metamodels will be combined to create an integral repository structure that at minimum supports all aspect models. The integral metamodel is the base for implementing a tool (chapter 5) that supports the graphical and textual representation of DEMO models. This repository can be used to create the model in organisations (chapter 6), automation and e.g. transform the PM to a Petri-net [85, Ch4] representation for validation of the PM, and other transformations that are possible due to this repository.

2.5 The Notion of Metamodel

To be able to validate the metamodel of DEMO, we need a definition describing the requirements of the metamodel[2]. We will use the following definition of a metamodel [29]: "Metamodels define sets of valid models, facilitating their transformation, serialisation, and exchange.".
Analysing this definition, we have to define which models are valid. The metamodel of these valid models needs to be sufficiently complete to describe all sets of models that are allowed. Moreover, the metamodel needs to be restrictive enough to reject models that are not valid.
According to this definition, metamodels should facilitate the exchange of the models. Following this part of the definition means that all aspects of the model that are drawn,

[2]Though a broader discussion of the notion of metamodel is possible, it is of no interest of this thesis and likely requires the same amount of pages to discuss this properly.

noted, related, or specified need to be described in such a way that the exact representation can be reproduced with the specification.
The serialisation of a model allows for the exchange of model information with possibly a different syntax than the model is represented itself. This is needed to store and retrieve models and business rules to and from repositories. As mentioned in section 1.4.2 the metamodel also needs to facilitate the model exchange of a visual representation of the model.
Transformation of the models is not included in DEMO's base but is part of the current research project. Therefore, we must be able to specify the DEMO models in the metamodel well enough to enable the partial transformation of the model (e.g. from ontological to implementation level).
This usage of metamodels for prescriptive and descriptive modelling [30, Section 2.2] makes the resulting models stable. The models will be semantically and syntactically sound, making the perception of the model predictable.

2.6 The Notion of Partial Metamodel

Due to the complexity of the metamodel that we present we decided to create partial metamodels. Partials metamodels describe the metamodel from a specific viewpoint. In other words, we leave out a large portion of the complexity and focus on a single aspect of the metamodel.
In the thesis we distinguish five partial metamodels:

- ontological-metamodel

  The high level ontological metamodel is the metamodel that is closest to the metamodel that lists all ontological principles of DEMO models [25]. For example, ontologically, the CAR cannot be an initiator of a transaction kind because an underlying elementary actor role must be the initiator. Therefore, this property type does not exist in the high level ontological metamodel.

  In contrast with the high level ontological metamodel, the ontological metamodel contains all implementation attribute types and property types for the DEMO metamodel. The ontological metamodel also includes property types between the concepts.

- verification-metamodel

  The logical part of the metamodel is about internal verification of the models represented. These verification rules of the verification-metamodel will be addressed as a separate metamodel.

- visualisation-metamodel

  The visualisation of a model seems trivial. Nevertheless, we decided to build a visualisation-metamodel concerning all visualisations of complexes.

- exchange-metamodel

  The exchange-metamodel is build up of two major parts:

  - data-exchange-metamodel

    The concept of a data-exchange-metamodel is intertwined in the concept of the ontological metamodel. We describe the data-exchange-metamodel in terms that a computer can understand. We choose to represent the data-exchange-metamodel in XML Schema Definition (XSD), which is a commonly used technique.

  - visualisation-exchange-metamodel

    All element that have been described in the data-exchange-metamodel have a visualisation element in the exchange metamodel. The structure of the visualisation elements is similar to the data-exchange-metamodel structure. All visualisation elements reference the data elements in the same model. The visualisation elements add the visualisation attributes for the specific diagram to that information (e.g. size, location, waypoints).

2.7 Fact Diagrams

In order to understand the metamodels presented in this thesis, we will explain a few concepts which are used in de graphical notation of the metamodels. First, an entity type is a representation of a thing with distinct and independent existence. Those things can be tangible (e.g. car) or intangible (e.g. transaction). Next, a property type is a binary fact type that maps the domain entity type to the range entity type. In writing, one often calls the property type 'a relation'. Last, an attribute type is a quality or feature regarded as an inherent part of a thing. Using these types, one can describe things and their relation. To create new things, one needs to add the operations of specialisation, generalisation, and aggregation. Specialisation is the restriction of an entity type to a specific property of the thing described (e.g. an electric car). Generalisation is combining concepts to form a new concept (e.g. car and boat form transportation). Aggregation is the clustering of concepts as a Cartesian product (e.g. car $\times$ supplier). Using these concepts and operations, one can build metamodels that we use in this thesis.

2.8 The Notion of Enterprise-grade

The term "enterprise-grade" does not originate from science[3], but is a term commonly used in practice. As reported by practitioners, the term is used to differentiate consumer products from enterprise products [31]. In [32], Gartner defines it as: "*Enterprise-grade describes products that integrate into an infrastructure with a minimum of complexity*

[3]Even though we can not claim to have conducted an in-depth literature survey on the term "enterprise-grade", the papers we did find (through a basic google scholar search) left the definition implicit.

and offer transparent proxy support." In line with this, enterprise-grade (in the context of software applications in general) is associated by practitioners [31, 33, 34] to characteristics such as *productivity*, *security* (including e.g. encryption of data, data security, granular levels of user access, protection of data, compliance), *integration*, *administration*, *support*, and *scalability*.

For a modelling tool to be enterprise-grade, it also implies that there must be a solid and secure repository (to store the models). Essential qualities for such repositories include *extensibility*, *maintainability*, *interoperability*, and *portability* [35]. Since enterprise engineering and architecting efforts typically involve many different aspects, as well as many different stakeholders [36, 37], it is important for modelling tools to provide different visualisations. Even more, these visualisations should be in a format that can be easily integrated into standard presentation and text editing software.

Finally, as often also included in evaluations by e.g. Gartner [32], an enterprise-grade tool needs to be backed by a company or organisation that can provide reliable after-sales service and maintenance. This also presupposes a committed and serious tool vendor, while the tool needs to be built using mature technologies (which certainly does not automatically imply proprietary technologies).

In conclusion, "enterprise-grade" primarily concerns non-functional quality criteria of both the tool, and the technology vendor providing the tool and/or associated services.

2.9 The Notion of Tool

The Cambridge dictionary defines a tool as "a program or feature of a program that helps you do particular things on a computer". This is a very broad definition and we need to narrow it down. For modelling one needs to:

- create elements and relationships between the elements
- (re)use elements and relationships for visualisation in some sort
- store the model and retrieve it later
- delete elements and relationships

In this thesis we define a automated modelling tool as a program that features the modelling activities like create, (re)use, delete, visualise, exchange, and store for elements and relationships between those elements.

Chapter 3

The DEMO Specification Language

Part of this chapter was previously published in a paper for the Enterprise Engineering Working Conference (EEWC) 2018 [2].

3.1 Modelling (in) DEMOSL

The DEMOSL is the specification language of (1) how DEMO models should be constructed, (2) what rules they must comply with, and (3) what information they should contain, and (4) how the model can be represented. Every DEMO model should comply with the DEMOSL and the DEMOSL should be able to define all valid DEMO models. Before building the whole metamodel of DEMOSL in Sparx Enterprise Architect (SEA) (section 5.2), we analysed consistency of the current metamodel in our research. Gouveia and Aveiro [38] verified the FM and suggested to make changes in the value type, but those did not affect the ontological-metamodel. Moreover, published literature mentions no changes in the metamodel. Furthermore, literature research reveals no documented, complete implementation of DEMOSL. Therefore, we conjecture that the metamodels have never been verified for automation. In the next paragraphs, we will analyse the DEMOSL metamodels and suggest changes where applicable.

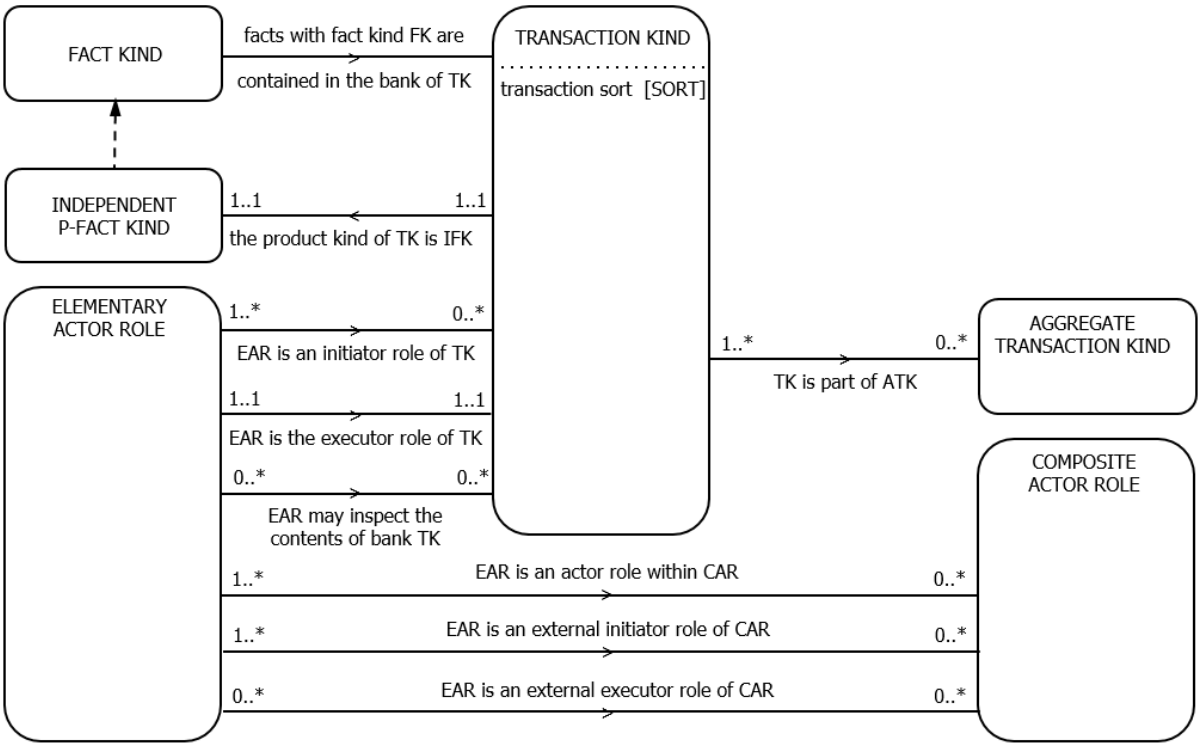


Figure 3.1: Diagram of the metamodel of the CM

The diagram of the ontological-metamodel of the Construction Model (CM), as shown in fig. 3.1, shows six entity types and represents the meta-level to combine actor roles and transaction kinds. The TK entity type models the transaction kinds, and ATK is the aggregation of transaction kinds. By the 'is part of' property type, a single component can visually replace the transactions in that scope. All created facts resulting from all transactions are contained within the Fact Kind and Independent P-Fact Kind. Interstriction between transaction kinds and actor roles are equivalent to the 'may inspect the contents of bank' property type. The entity types EAR and CAR contain the actor roles of the model. A SoI hierarchy of actor roles is available through the 'is an actor role within' property type. The elementary and the composite actor roles both have the initiator and executor roles modelled as the property types indicate.

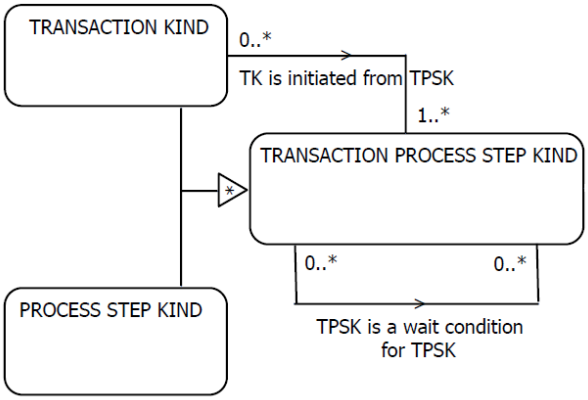


Figure 3.2: Process Metamodel

The PM metamodel as illustrated in fig. 3.2 shows three entity types. The process-step-kind-entity-types aggregate the transaction kind with the process step using a Cartesian product. This aggregation allows for modelling the relationship between a process step kind of one transaction kind and a request step in another transaction kind using the 'is initiated from' property type. Also, the wait conditions between two process step kinds are equivalent to the 'is a wait condition for' property type. The transaction kind entity type is the same as the transaction kind entity type in the CM metamodel.

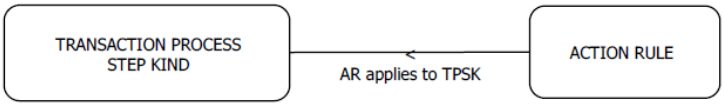


Figure 3.3: Action Metamodel

The AM metamodel as illustrated in fig. 3.3 has a single entity type representing action rule information of a specific process step. In the formal specifications, the AM is specified in Extended Backus - Naur Form (EBNF) [40]. The EBNF specification mentions three main parts. The first part specifies the preconditions for an execution of the action. The second part specifies the evaluation of the conditions. The last part specifies the executed actions in case of a valid or invalid condition. The relations between the AM and the FM appear to be implicit.

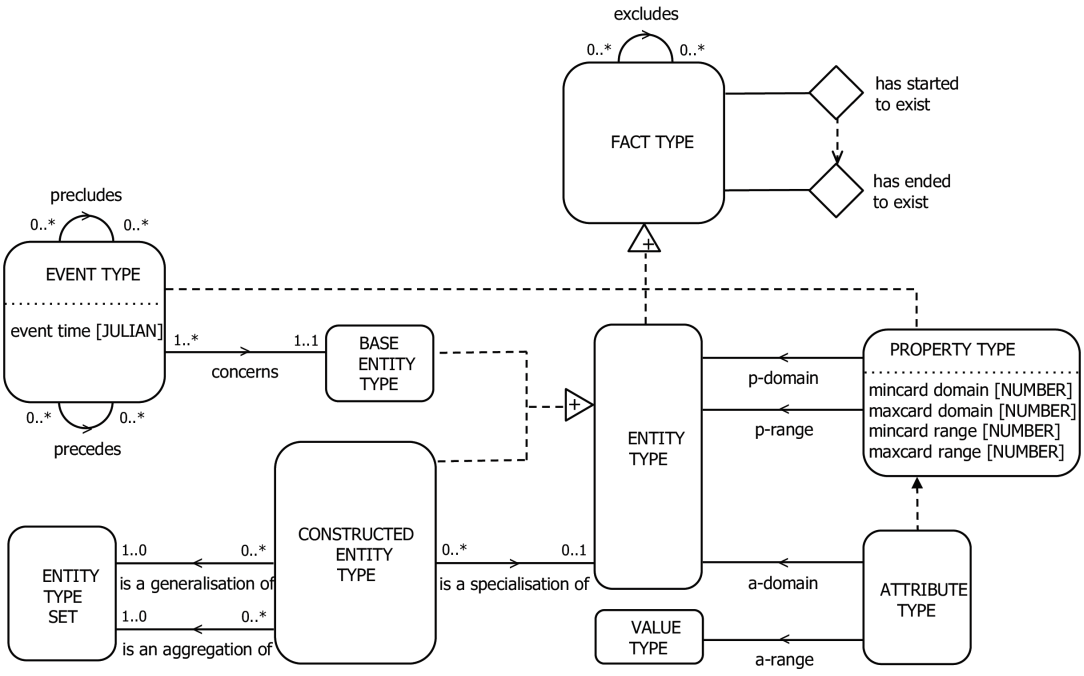


Figure 3.4: Fact Metamodel

The FM metamodel as illustrated in fig. 3.4 contains eight entity types. This ontological-metamodel (1) allows the creation of simple entity types and specialised, generalised or aggregated entity types. Entities can contain two types of attributes (2): value type and properties related to other entity types. An event type property (3) stores occurring events, similar to the property type, within an entity type. The combination of the event type, entity type, property type, value type, and attribute type entity types results to the generalisation fact type (4) which can start or end to exist (wherein DEMO facts only start to exist).
With the partial DEMOSL ontological-metamodels in mind, we tried to project existing DEMO models onto the ontological-metamodel and found some practical imperfections.

3.2 Observations

With the partial DEMOSL-metamodels in mind, we tried to project the existing DEMO models onto the metamodel using a newly developed automation tool. In doing so, we discovered some practical imperfections on which we elaborate in the next sections. Also, during the project of building an automation tool to support DEMO modelling, the analysis of the ontological-metamodel showed some inconsistencies. We found that not all partial metamodels allow to produce correct DEMO models. Moreover, not all parts of the DEMOSL metamodel were connected. This may have consequences for the usability of DEMO models as a whole. Theoretical benefits might not be usable when one reduces the DEMO model to its aspect models. DEMO claims to be a methodology that can reveal the essence of an organisation by combining the four partial analyses. A search within

the existing literature did not yield any research or scientific findings on this subject.

3.3 DEMOSL Inconsistencies and Omissions

The consistency of the current DEMOSL metamodel of DEMO was analysed with automated tools. Earlier, Gouveia and Aveiro [38] verified the FM and suggested to make changes in the value type but those did not affect the metamodel. Kervel [39] has created his own model to specifically describe the transaction pattern of DEMO models. Other literature research did not yield any results on suggested or applicable changes to the metamodel. Furthermore, literature research did not reveal any documented or published complete implementation of the DEMOSL metamodel. The DEMO community, which consist of academic and practical specialists, also does not have any research available on this subject.
In the next paragraphs, we will further elaborate on the analysis of the DEMOSL metamodels. We used the published DEMOSL versions 3.6 [27] and 3.7 [1] for the analysis of the metamodel. DEMOSL 3.7 has been published and some results of this research project are inclused already..

3.3.1 DEMOSL Metamodel

We can make three observations due to the exchange-metamodel requirement that arises from section 2.5. Firstly, no part of the DEMOSL metamodel is currently capable of capturing all necessary information to exchange names, formulations, or visual information from diagrams, tables, or specifications. In the past, no one had the intended use of DEMOSL for automated tools and, therefore, did not request to describe these aspects of the metamodel. The current version of DEMOSL is only describing the high-level ontological structure. Secondly, although ontology perspective considers the name of an object as the identity of that object, the ontological-metamodel must include the name to be able to exchange this identity from a modelling perspective. Finally, concerning the diagrams and tables used in DEMO, no visualisation-metamodel is given with DEMOSL, though some visualisation examples are present and imply a visualisation-metamodel. This definition is needed to be able to exchange the exact visual information.

3.3.2 Construction Metamodel

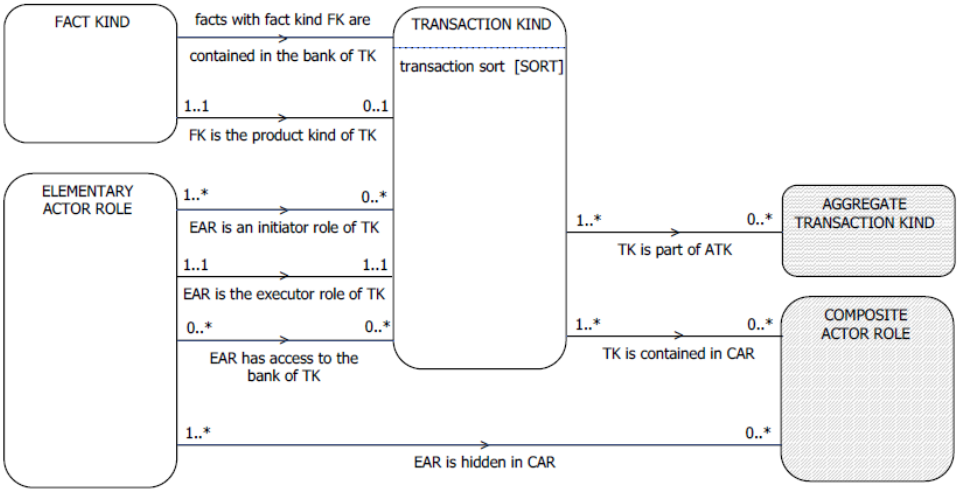


Figure 3.5: Metamodel 3.6 of the CM [27, sl.27]

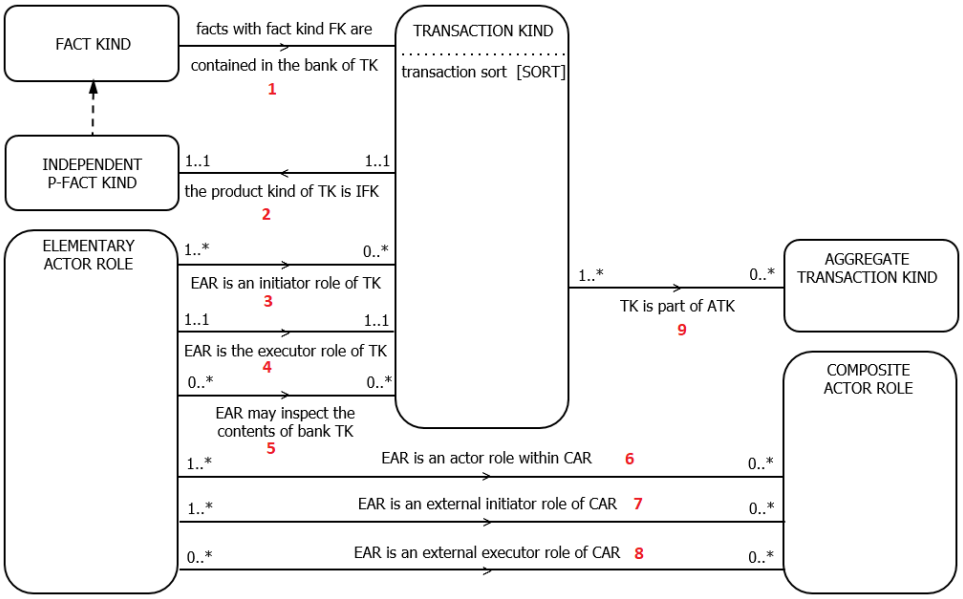


Figure 3.6: Metamodel 3.7 of the CM [1, sl.31]

The metamodel of the CM 3.6 (fig. 3.5) [27, sl.27] shows five entity types and represents the meta-level to combine actor roles and transaction kinds. In addition, the metamodel of the CM 3.7 (fig. 3.6) [1, sl.31] shows six entity types. The observed change in the metamodel is a partial result of the research we conducted. We will further elaborate on the suggested improvements and will highlight the remaining issues.
The TK entity type represents Transaction Kinds (TKs). ATK is the aggregation of multiple TKs. This concept comprises the aggregation of all information of individual TKs. The 'TK is part of ATK' property type, fig. 3.6(9), therefore, allows for replacing the transactions within a selected set of TKs by a single component, an ATK. In this

way, the ATK can visualise the involved TKs in an aggregated form. ATKs that are out of the SoI have no TK to refer to. The ontological-metamodel states that this reference is mandatory (1..*), whereas an example [26, p.89] shows several unreferenced instances of ATKs. Because DEMOSL does not specify another representation to register this property type it is either an omission in the diagram or an inconsistency within DEMOSL.
The Fact Kind (FK) entity type in DEMOSL 3.6 represents both the coordination fact and production fact of a TK. This notion is not consistent with the FM; therefore, it has been changed in DEMOSL 3.7. The FK and Independent P-Fact Kind (IFK) now contain all transactions created facts.
The 'EAR has access to the bank of TK' that we have rephrased to 'may inspect the contents of bank' property type, as shown in fig. 3.6(5), models the interstriction between transaction kinds and actor roles. Solely inspection of the ATKs cannot be represented in the ontological-metamodel.

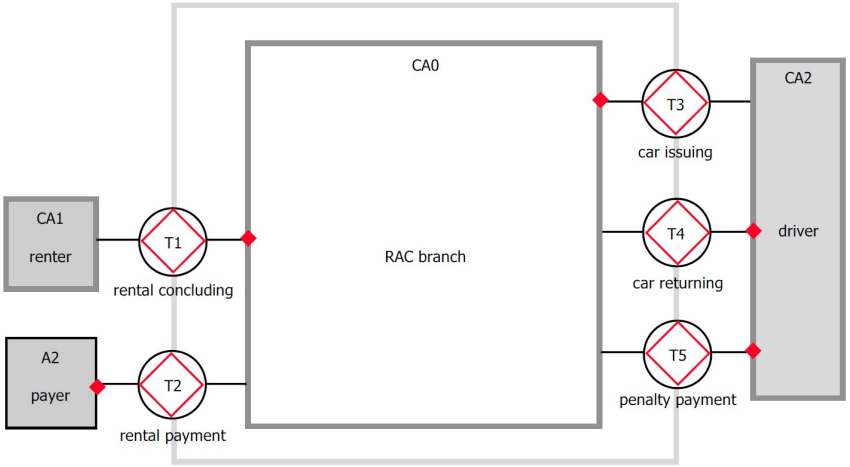


Figure 3.7: RAC Model from TEoO [26, p.72]

As can be seen in the diagram shown in fig. 3.7, a grey boundary is displayed. This boundary has been described in DEMOSL 3.6 [27, sl.7] and 3.7 [1, sl.7] as the SoI. The DEMO methodology states that the TK can be on the boundary of the SoI to represent a TK communication between the inside and outside of the border of the SoI. Unfortunately, the DEMOSL 3.7 ontological-metamodel cannot relate any component as being inside or outside any SoI, nor can it model the name of the SoI.
In the entity types EAR and CAR the actor roles of the model are stored.

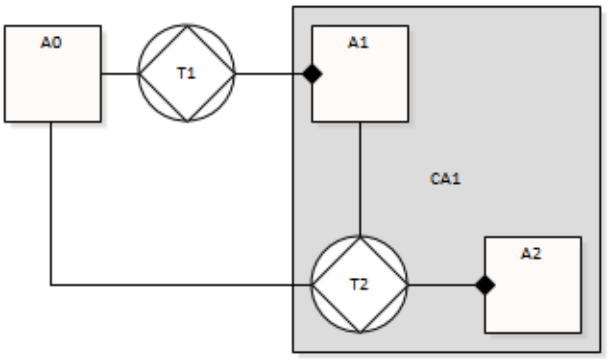


Figure 3.8: Transaction Kind and CAR

The initiator and executor roles are described separately as property types for the elementary and the composite actor roles. This set of property types has evolved from the DEMOSL 3.6 metamodel where 'EAR is hidden in CAR' was not sufficient for modelling various EAR roles within a CAR. A collection of actor roles is available through the property type 'EAR is an actor role within CAR', fig. 3.6(6). This property type requires that for every CAR at least one EAR is present. When modelling an actor role outside the SoI, this property type's cardinality cannot hold because the associated EAR does not belong to the model.
Although the CAR includes TKs according to [1, sl.31'note 4'], the property type 'TK is contained in CAR' of version 3.6 has been omitted. Therefore, no explicit property type from the transaction to a CAR exists. Moreover, [1, sl.31'note 4'] states the inclusion of the TKs between the EARs within the CAR. The situation of CAR2 in fig. 3.9 and CA1 in fig. 3.8 shows that the metamodel is not able to handle this situation. In the latter, A0 would initiate CA1. Without this 'TK is contained in CAR' ontological-metamodel property type, it is not clear whether the transaction kind T2 in fig. 3.8 is part of CAR.

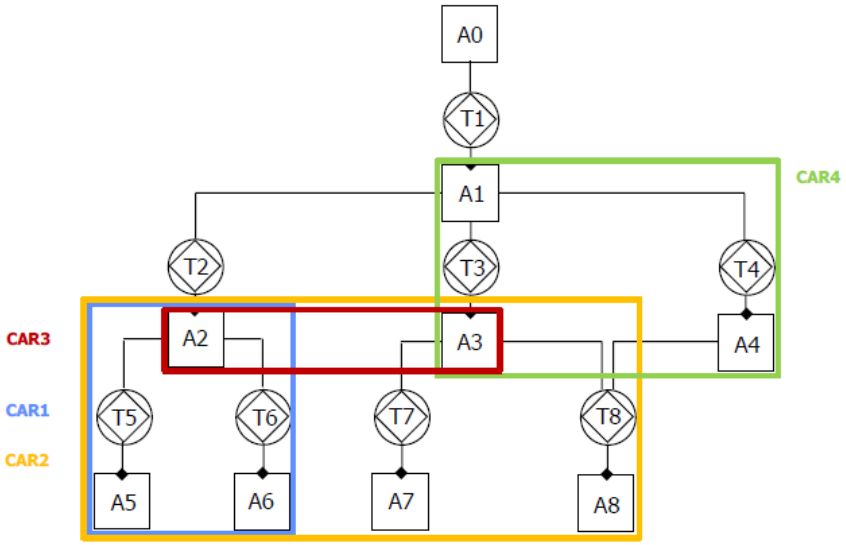


Figure 3.9: CAR example [1, sl.10]

In the DEMO methodology, the first step is to create a CM and to create a CAR to relate it to the border transactions. The metamodel cannot capture these CARs with their border transactions.
The next step in the DEMO methodology is to reveal the inner actor roles within the CAR. When these actor roles are too complex to be revealed at once, one uses new CARs where more actor roles can be revealed. Neither the ontological-metamodel of DEMOSL 3.6 nor 3.7 can model the hierarchical CAR. When an EAR is part of multiple CARs, the CARs might relate in some way. This property type is also not present in the metamodel. In the metamodel of 3.7, an example as shown in fig. 3.9 of multiple CARs within the same CM has been given. CAR1, as well as CAR3, is contained within CAR2[1] but whether this property type is relevant cannot be modelled.
A summary of the issues found in the CM:
CM.a inconsistent property type to the FM(but this has been fixed in DEMOSL 3.7);
CM.b mandatory TK for ATKs;
CM.c mandatory EAR for CARs;
CM.d missing inspection property type to separate ATKs;
CM.e missing scope of interest;
CM.f missing CAR to TK property type;
CM.g missing CAR hierarchy;
CM.h missing property type TK in CAR;
CM.i missing interstriction between ATK and CAR or SoI.

3.3.3 Process Metamodel

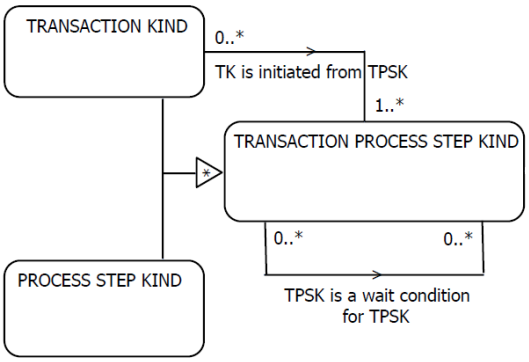


Figure 3.10: PM Metamodel 3.6 [27, sl.28]

[1]The initiator and executor roles in the example notes are incorrect

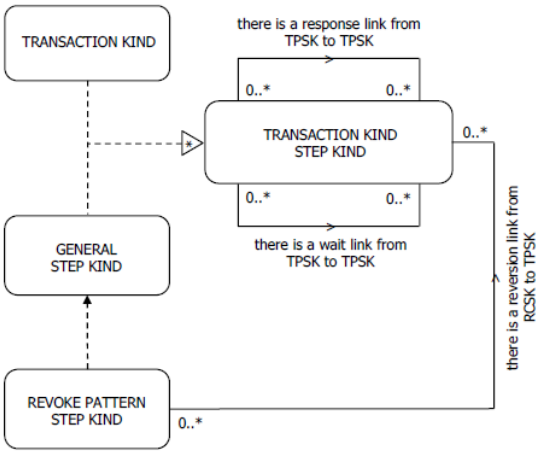


Figure 3.11: PM Metamodel 3.7 [1, sl.32]

The PM ontological-metamodel of DEMOSL 3.6 (fig. 3.10) shows three entity types. The PM ontological-metamodel of DEMOSL 3.7 (fig. 3.11) shows four entity types. The entity type TPSK is partially renamed in version 3.7 to Transaction Kind Step Kind (TKSK), but we will still refer to this entity type with TPSK. The Transaction Kind (TK) entity type represents the same entity type as shown in the CM (fig. 3.6), and creates the connection between the CM and the PM.
The Process Step Kind (PSK) entity types in the metamodel of DEMOSL 3.6 extend the TK with the process steps using a Cartesian product. This extension allows for modelling the relationship between a TPSK of one TK and a PSK in another TK using the 'is initiated from' property type. Moreover, the 'is a wait condition for' property type represent wait conditions between two process steps.
The property type 'TK is initiated from TPSK' in the ontological-metamodel of DEMOSL 3.6 did not allow a TPSK to invoke a Revoke Step Kind (RSK) in another TK. Therefore, in metamodel of DEMOSL 3.7, the property type is changed to a self-reference: 'there is a response link from TPSK to TPSK'. This property type does not sufficiently restrict the model and needs [1, sl.32'note 2'] that the transaction pattern will limit the possibilities of linking the appropriate steps.
The transaction pattern itself has been extended with a reversion link to support the step kinds from Revoke Pattern Step Kind (RPSK) patterns to the General Step Kinds (GSK). The reversion link is part of the transaction pattern. The metamodel should include all steps of the internal transaction pattern. However, the ontological-metamodel does not allow for modelling the complete transaction pattern.

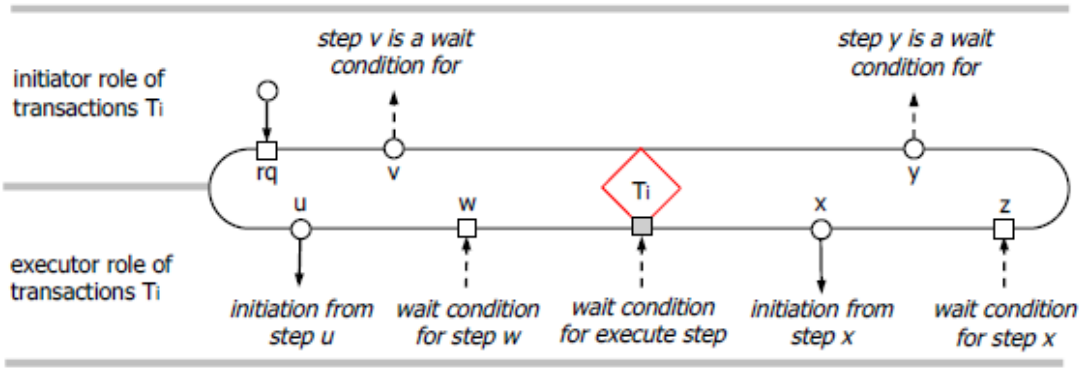


Figure 3.12: Process Model Legend [1, sl.18]

In the PSD visualisation (fig. 3.12), swim lanes are introduced in the diagrams. These swim lanes are not present in the ontological-metamodel.
In summary, the issues found in the PM metamodel are:
PM.a links between TPSK instances are insufficient (but these are fixed in 3.7);
PM.b only partially modelling of the complete transaction pattern is possible;
PM.c swim lanes are not present in the ontological-metamodel.

3.3.4 Fact Metamodel

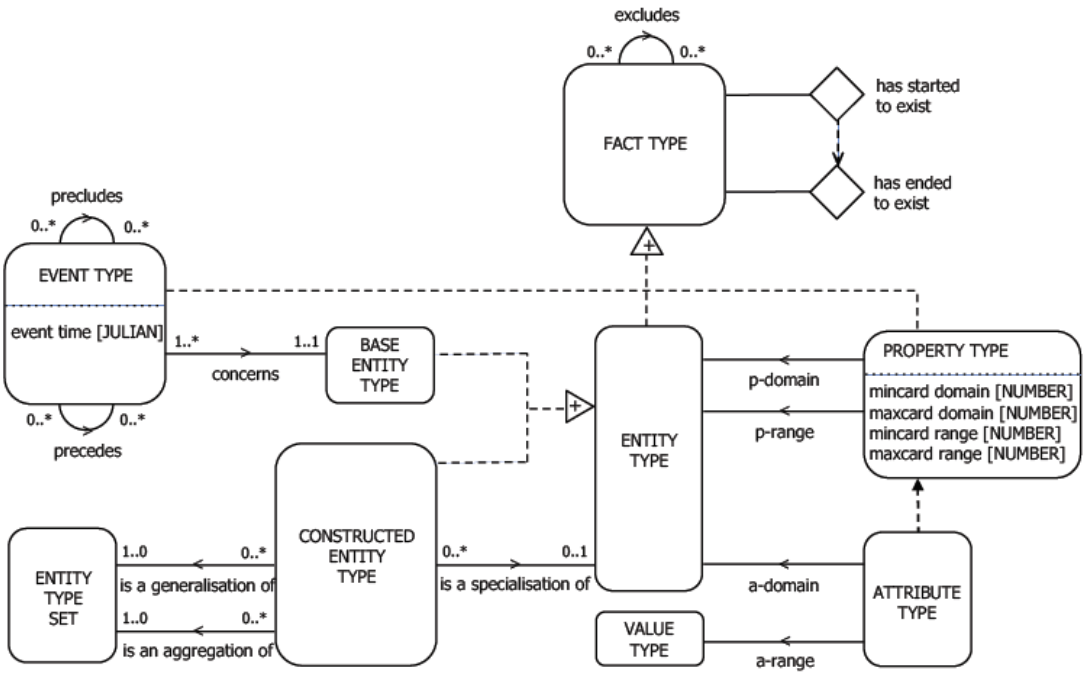


Figure 3.13: Fact Model 3.7 Metamodel [1, sl.34]

The FM diagram and the ontological-metamodel are expressed in the same visualisation. Therefore, the metamodel of the FM is illustrated in **bold** and we have used the *italic*

font for the FM diagram itself. Furthermore, the FM metamodel 3.7 as shown in fig. 3.13 contains eight **entity type**s. The difference between the DEMOSL 3.6 and the 3.7 version is the addition of the 'P-' naming prefix ('P' stands for **P**roduction-world). We omit this P-naming prefix in this section for short writing where no ambiguity can be found.
We will briefly explain the FM metamodel and provide comment on the things we see in this diagram. This FM metamodel allows for simple *entity type*s but also for specialised, generalised or aggregated *entity type*s. *Entity types* can contain two types of *property type*s: *value type*s (i.e. attribute) and *property type*s (i.e. relation).
An *entity type* can be modelled using the concept **entity type**. The name of a specific *entity type* [1, sl.22] has not been added to the FM metamodel.
The *property type*, or relationship, between *entity type*s is expressed as a **property type** that consist of a domain **entity type** and a range **entity type**. The name of a *property type* [1, sl.22] has not been added to the FM metamodel.
The **attribute type** is a specialisation of a **property type** and acts the same way as a **property type**. [1, sl.34-'footnote 2'] states that the property types 'adomain' and 'arange' are also specialisations. The name, dimension, and unit of an *attribute type* [1, sl.22] has not been added to the FM metamodel.
Every **property type** has a minimum and maximum cardinality for the domain and range.
The **event type** entity type matches the IFK entity type of the CM. Therefore, it adds two property types to the creation of a *fact kind* in the CM: precedes and precludes. The precedence law [1, sl.26] states that two **fact kind**s as have an order in the time line. The second **fact kind** can only occur when the first one has occurred. The preclusion law states that the two **fact kind**s can only occur both if the first **fact kind** has not occurred before the other. The above mentioned precedence law, as well as the preclusion law, do affect the CM and the PM. Neither the metamodel nor the diagrams of the CM or the PM mention this. The *property type* event time on the **event type** entity type represents the moment in time line the event occurred. This event time duplicates the 'has started to exist' event type.
The 'concerns' property type does link the **event type** to the **base entity type** in such a way that every transaction kind that creates an independent fact kind also needs to have a concerning **entity type**.
This notion of concerns to a **base entity type** means that an **event type** can never concern a **constructed entity type**. Therefore, transaction kinds cannot be a concern to specialisations of entity types. In the example RAC [26, p.75] is shown that the 'RENT-PAID RENTAL' is a specialisation of 'RENTAL' while 'RENT-PAID RENTAL' is the concern of 'P2 the rent of Rental is paid'.
The **constructed entity type** deals with the specialisation of **entity type**s and the generalisation and aggregation of **entity typeset**s. The distinction between entity type and entity typeset connects together the entity types used for the set operation. [1, sl.34-'footnote 3'] does nonvisually connect the **entity typeset** to the **entity type**. We only find a typo '1..0' in the property type of the **constructed entity type** in the diagram.
The **fact type** is a generalisation of an **entity type** which can start or end to exist. To be able to model the existence of a fact, we will need the time it came info existence and

the time its existence ended. Occurring production events will be stored as an **event type**, similar to the **property type**, and related to an **entity type**.
The exclusion rule on a **fact type** [1, sl.25] can be used to make two disjoint collections of **entity type**s, **property type**s, or even **event type**s.
In summary the issues we found in the FM metamodel are:
FM.a name on entity type is missing;
FM.b name on property type is missing;
FM.c name, dimension and unit on attribute type are missing;
FM.d CM and PM lack precedence and preclusion laws;
FM.e specialisations cannot be concerned with event type;
FM.f cardinality on entity typeset is incorrect;
FM.g no time in start and end events.

3.3.5 Action Metamodel

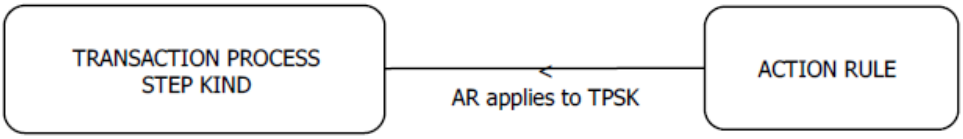


Figure 3.14: Action Metamodel 3.7 [1, sl.33]

The AM metamodel as illustrated in fig. 3.14 [1, sl.33] has a single entity type representing action rule information of a specific PSK with a reference to the related TPSK. In the formal DEMOSL specifications, the AM is specified in EBNF [40]. The specification mentions three main parts: The first part specifies the preconditions to execute an action. The second part specifies the evaluated conditions of the action. The last part specifies the executed actions in case of a valid or invalid condition. The syntax of the property types to the FM is not fully specified. In the AM it is not clear whether the relationship between the product kind and entity type is about reading, writing, or creating a FK.
The information in the AM is not sufficiently detailed to verify a model. We can specify the verbalisation used in the Action Rules Specification (ARS) as property types.
We analysed the ARS with ANother Tool for Language Recognition (ANTLR) to verify the completeness of the specifications of action rules. Although ANTLR does have a slightly different syntax on EBNF, we translated all rules which need to be verified.

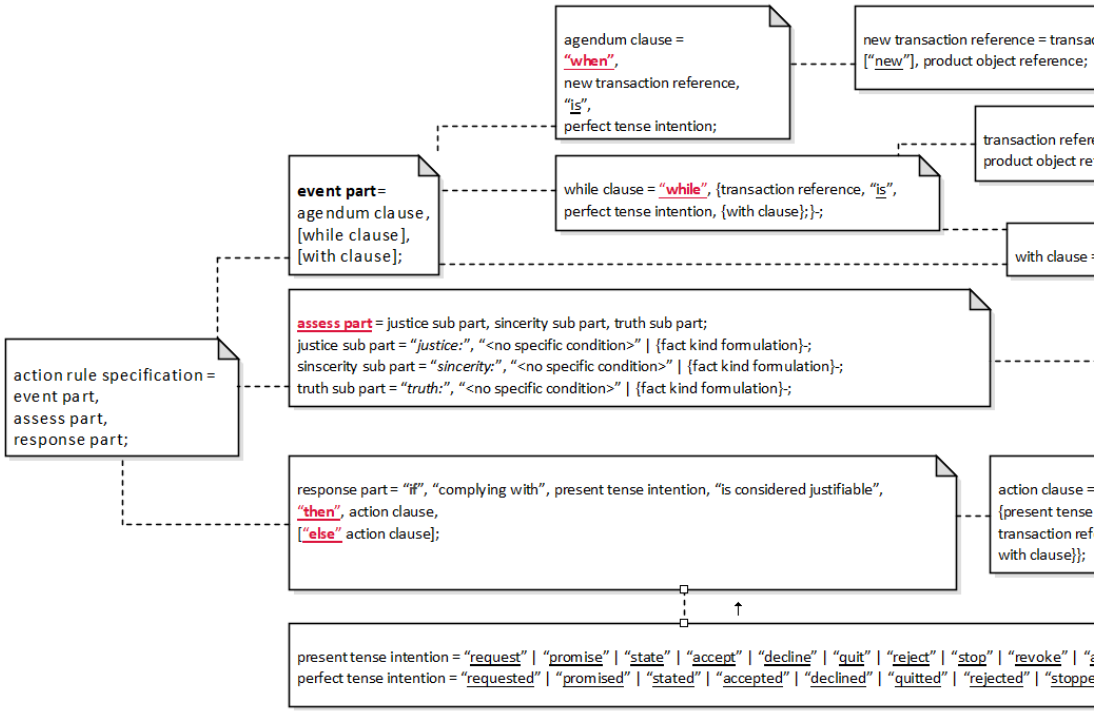


Figure 3.15: (Partial) Action Rule Specification

The result from our translation is that the following specifications were missing:

- property kind name;
- object variable;
- dimension;
- perfect tense sentence;
- product object reference.

The above mentioned specifications need to be defined, or their reference needs to be altered, in order to be able to use the ARS. Furthermore, we found that some definitions of variables were not distinct enough to be parsed by the ANTLR specifications. This missing distinction could mean that the specification is ambiguous in the variable definition. The attribute variable, abstract variable and product variable form connections to the fact model. These connections should be present in the metamodel as illustrated in fig. 3.13.
In summary, the issues found in the AM metamodel are:
AM.a specifications of action rules need additions and elaborations;
AM.b naming of attribute, abstract and product variable needs to be added.

3.3.6 Visualisation

Since the DEMOSL [1] primarily focuses on consistency and completeness of DEMO models from a "content" perspective. It is about educational examples in the most likely situations. DEMOSL does not include any specifics about the actual representation of these models in terms of diagrams, tables and other possible visualisations. As such, the DEMOSL does not provide guidelines regarding the concrete syntax of models in

terms of e.g. shapes and icons to be used. When examining a corpus of models produced across different cases, we found various variations in drawings of elements that each could be interpreted as the convention of those elements. In moving towards (standardised) tool support, this had to be remedied in terms of an explicit metamodel of the allowed visualisations.

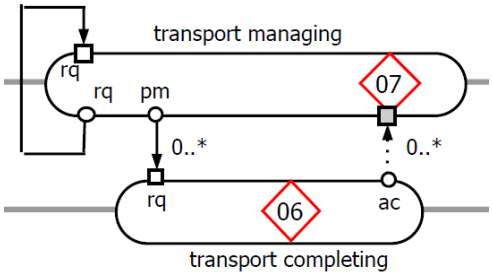


Figure 3.16: PSD specification example

As an example, consider the diagram provided in fig. 7.3. In an OCD, the shape is normally drawn as a circle enclosing a diamond and this circle is stretched in the PSD with the diamond displayed at a seemingly random position within this "stretched circle". In fig. 7.3, the name of the transaction appears below or above or at the side-top of the transaction fig. 7.3, the diamond is at a certain percentage from the left side, the stretching is a random length, while the swim lane usage is not consistent (i.e. 07 has the same initiator and executor but is visualised on top of two swim lanes).
Furthermore, the examples of the same case are different between different DEMO versions without explaining why this is the case, making it impossible to give guidance to readers that were trained in a different version to interpret the visualisation, and also making it impossible for a tool to comply to the specifications.

3.3.7 Summary

When we summarise our findings in the DEMOSL we will get a list of inconsistencies and omissions that need to be solved in order to support automated modelling in DEMO. In the next chapters we will make enhancements to the DEMOSL for this purpose and implement a automation tool which contains the metamodel and will support automated modelling in DEMO.

Chapter 4

The DEMO Specification Language Proposal

Parts of this chapter have been previously published in the paper for the OTM/FBM 2018 conference [3].
Originally, the DEMO Specification Language (DEMOSL) was designed to better understand the concepts that need to be included in DEMO models. However, for the development of an automated tool support for DEMO, more details and specificity of implementation are needed than what is included in the original DEMOSL. These details include the graphical layout of diagrams, exchange of models between tool environments, as well as the specificity of the actual models. As such, the goal is to create an improved version of the existing DEMOSL that is complete enough to automatically verify DEMO modelling rules and restrictions using automated tools. Though a aim for completeness would be ideal, it is not feasible within this research.
The resulting specification of DEMO is divided into four kinds of models (i.e. ontological, visualisation, exchange, and visualisation exchange), and four associated metamodels. These four model kinds need to comply with the requirements as stated in [6].
This chapter has been called a proposal and part of the findings from our research as described in this chapter is meant to be incorporated in a successor of the DEMOSL version 3.7. In this chapter we will explain some models and metamodels in the context of DEMOSL. We have created fig. 4.3 to clarify the relationships and transitions between these metamodels and models. To make fig. 4.3 a bit more comprehensive and readable we have built-up the figure in three layers. The bottom layer is the MU-theory, as explained in section 2.3. The layer on top of the MU-theory is the exchange layer as shown in fig. 4.1. This layer is modelled conform to the MU-theory and is dedicated to the ontological-(meta)model and the exchange-(meta)model. The top layer is the visualisation layer as shown in fig. 4.2. These two layers on top of the MU-theory are related in a presenting and referencing relationship.
These metamodels as shown in fig. 4.3 add information, structure, and completeness to the existing version of DEMOSL. Our research concluded that without implementation of these improvements the automatic verification and exchange of DEMO models using automated tooling is not possible. Figure 4.3 shows the landscape of metamodels that

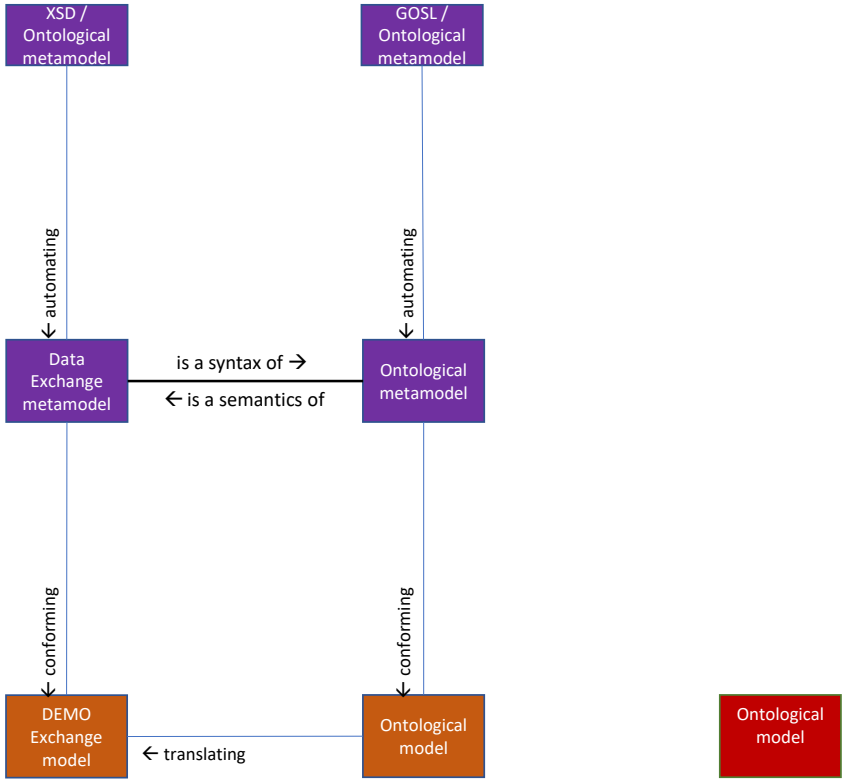


Figure 4.1: the exchange subset of the metamodel

together have the capability of modelling organisations using DEMO. The landscape is visualised as a three dimensional framework, projected on top of the framework from the MU-theory as described insection 2.3.
The two levels that have been projected on top of the framework from the MU-theory correspond to the distinction between language and visualisation perspectives as discussed in section 2.2.
The language perspective part is implemented in terms of automation software. This is needed in order to make it possible to reason about (1) the (ontological) model itself, (2) the way it may be exchanged, as well as (3) how these are visualised.
The visualisation perspective is concerned with the way how an actual DEMO model has to be visualised in terms of diagrams. This leads to a visualisation model conform to a visualisation metamodel. A visualisation model involves a set of visualisation attributes (position, size, colours, etc) on top of an (visualised part of) ontological model. The visualisation metamodel describes these attributes and contains the scripting and data structure of the diagrams. Restrictions on the allowed elements, attributes, and connections in a diagram are also formalised in the visualisation metamodel.
We have made improvements to the metamodel of DEMOSL 3.7 [1] according to the findings as described in chapter 3 (fig. 4.5). We have not only expanded the ontological-metamodel with some extra information, but we have also updated the ontological-metamodel, the verification rules, the visualisation-metamodel, and the exchange-

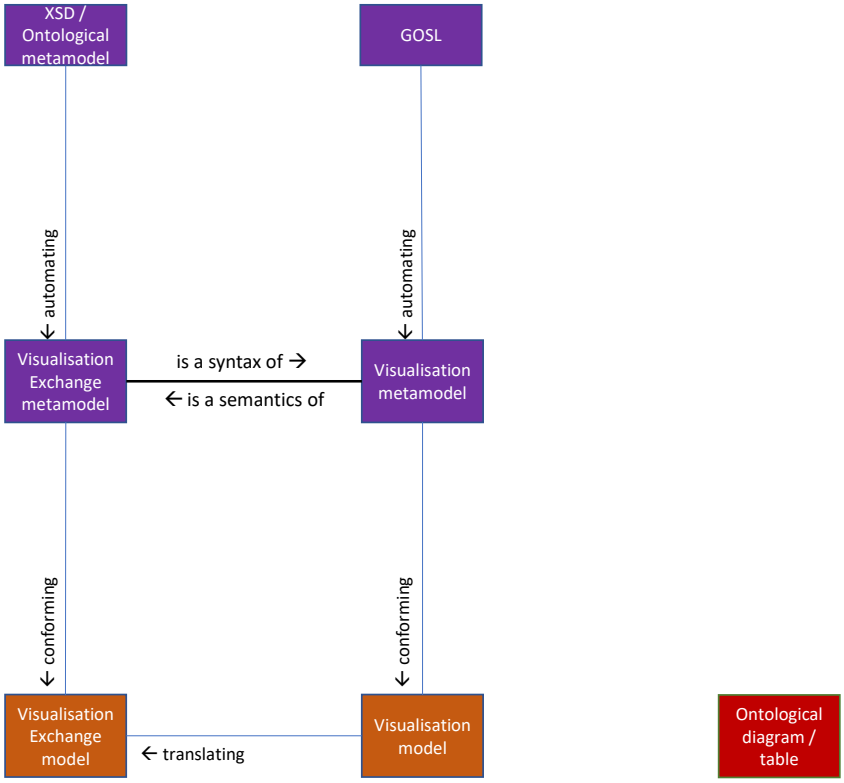


Figure 4.2: the visualisation subset of the metamodel

metamodel.
When we combine all partial ontological-metamodels from DEMOSL and extend this metamodel with the extra information, this results in the ontological-metamodel as shown in fig. 4.5. This ontological-metamodel includes all the properties we want to store for the concepts of DEMO but does not involve the diagram metamodel. Therefore, it is an automation-oriented ontological-metamodel.
Subsequently, for every component of the high level metamodel we need to define the ontological-metamodel, verification rules, exchange metamodel (XSD), programming model, visualisation model, and examples for clarification. For every diagram, there is a definition available for the set of entity and property types called the diagram metamodel. In general, modelling techniques allow for the representation of different properties of the object that is being modelled. Only a combination of these models will come close to the representation of the whole object. Therefore, for every component of the metamodels we need to define:

1. The ontological-metamodel for the item and its property types.
2. The verification-rules on the collections of items.
3. The exchange-metamodel of the objects.
   (a) XSD specification for the set of elements.
   (b) XSD specification for the item.

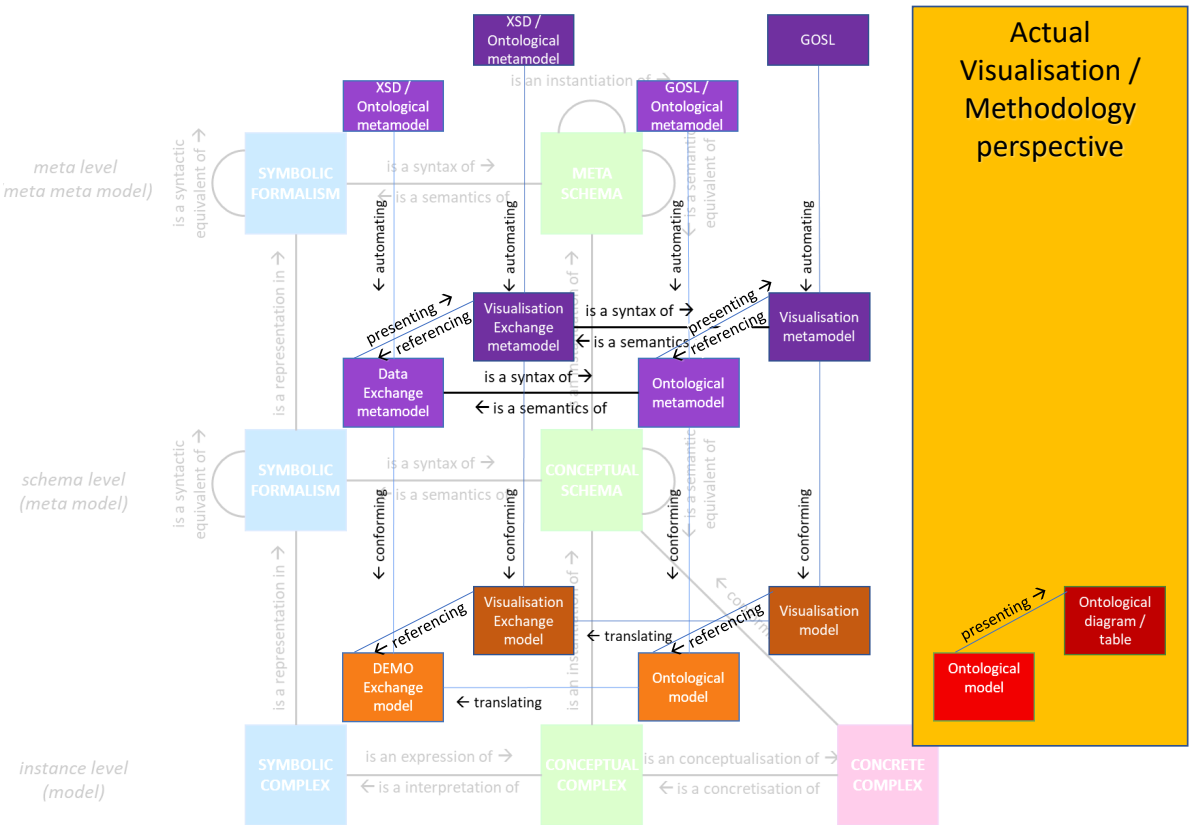


Figure 4.3: metamodel set. Combination of figs. 2.16, 4.1 and 4.2

(c) XSD specification for the type.

(d) XSD specification for the visual element.

(e) XSD specification for the visual connection.

4. A programming model to implement the verification rules and metamodel.
5. The visualisation model in SEA.
6. Provide an example in SEA.

With this list of models and associated metamodels, we observed that the representation of the DEMO metamodel is sufficiently complete to verify each DEMO model and exchange information to recreate the DEMO model. We re-modelled the ontological-metamodel according to our findings as described in [4] and chapter 3.
This chapter will cover items 1 to 3. Item 4 will be covered in section 5.3, item 5 will be covered in section 5.2 and finally item 6 will be covered in part II of this thesis.

4.1 Partial Metamodels

In the following sections, we discuss all mentioned models (i.e. ontological, visualisation, and (visualisation) exchange) and associated metamodels from right to left, from the top to the bottom as shown in Figure 4.3

4.1.1 Ontological-Metamodel

The high level ontological-metamodel is the metamodel that is closest to the metamodel that lists all ontological principles of DEMO models [25]. For example, from an ontological perspective, the CAR cannot be an initiator of a transaction kind because an underlying elementary actor role must always be the initiator. Therefore, this property type does not exist in the high level ontological-metamodel. The high-level ontological-metamodel of DEMO is drawn in fig. 4.4.
The diagram shows the existing property types and concepts in black. The newly added property types and concepts are highlighted in red. This high level ontological-metamodel, shown in fig. 4.4, is the base for the ontological-metamodel shown in fig. 4.5.

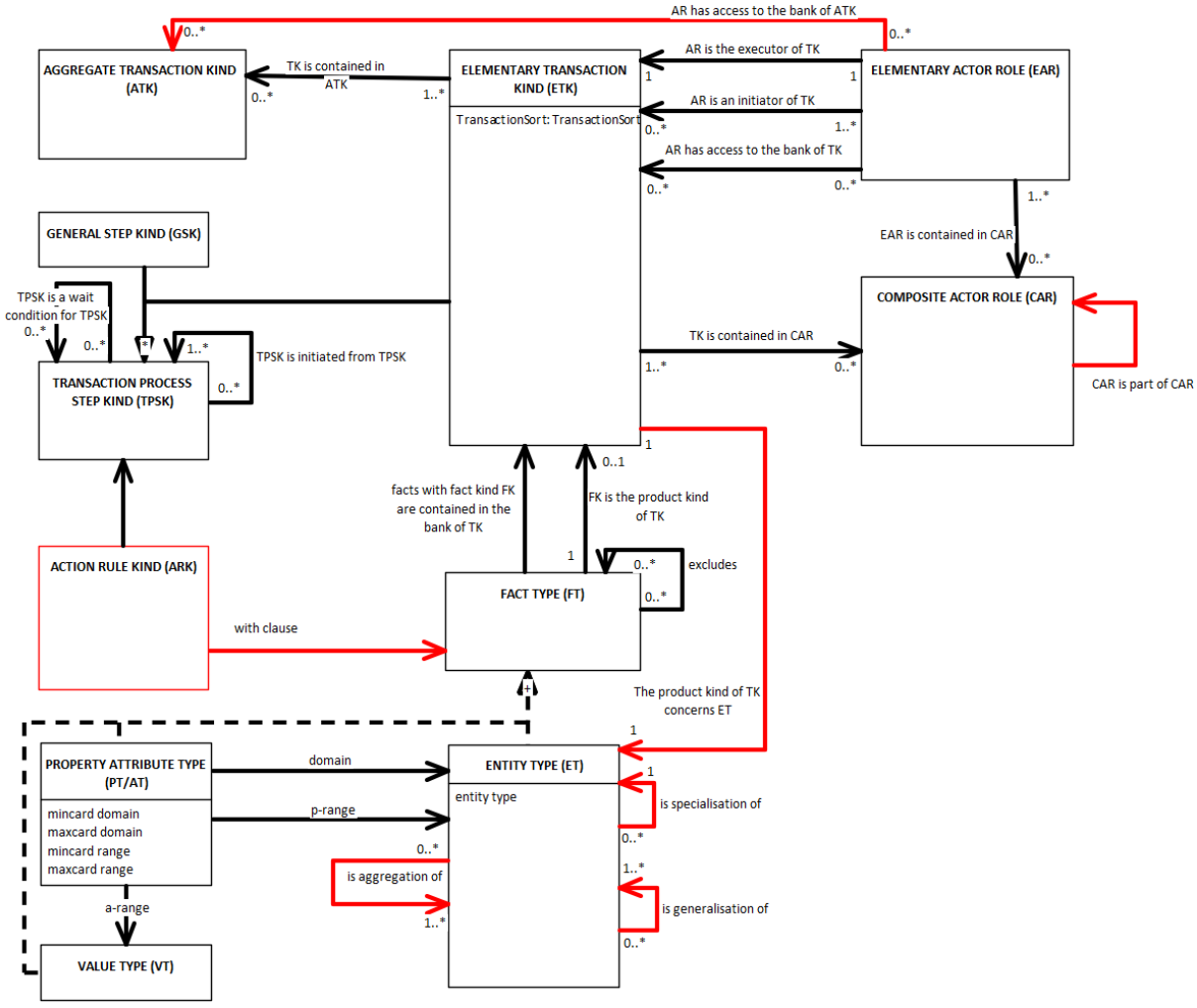


Figure 4.4: DEMOSL high-level ontological-metamodel

In contrast to the high-level ontological-metamodel, the ontological-metamodel below (see fig. 4.5) does contain all implementation attribute types and property types for the DEMO metamodel. This ontological-metamodel of DEMO (fig. 4.5) has property types between the concepts. In fig. 4.5, we did not visualise the removed property types, but these can

be found by comparing fig. 3.6, fig. 3.11, fig. 3.15, and fig. 3.13 to fig. 4.5. The example mentioned above concerning the differences in the metamodels involves the Composite Actor Role (CAR). If a CAR is an initiator, one should be able to represent it in the ontological-metamodel. Whenever one designs a CAR that initiates a transaction, this instantiation of this property type must be present in the DEMO model, and thus needs a representation in the ontological-metamodel. Therefore, the ontological-metamodel also has the property type 'AR is an initiator of TK' (as shown in fig. 4.5 on the top-right) from the CAR to the TK. Furthermore, verification rules have been designed to make sure only the correct property type instantiations can be present in the final ontological-metamodel.

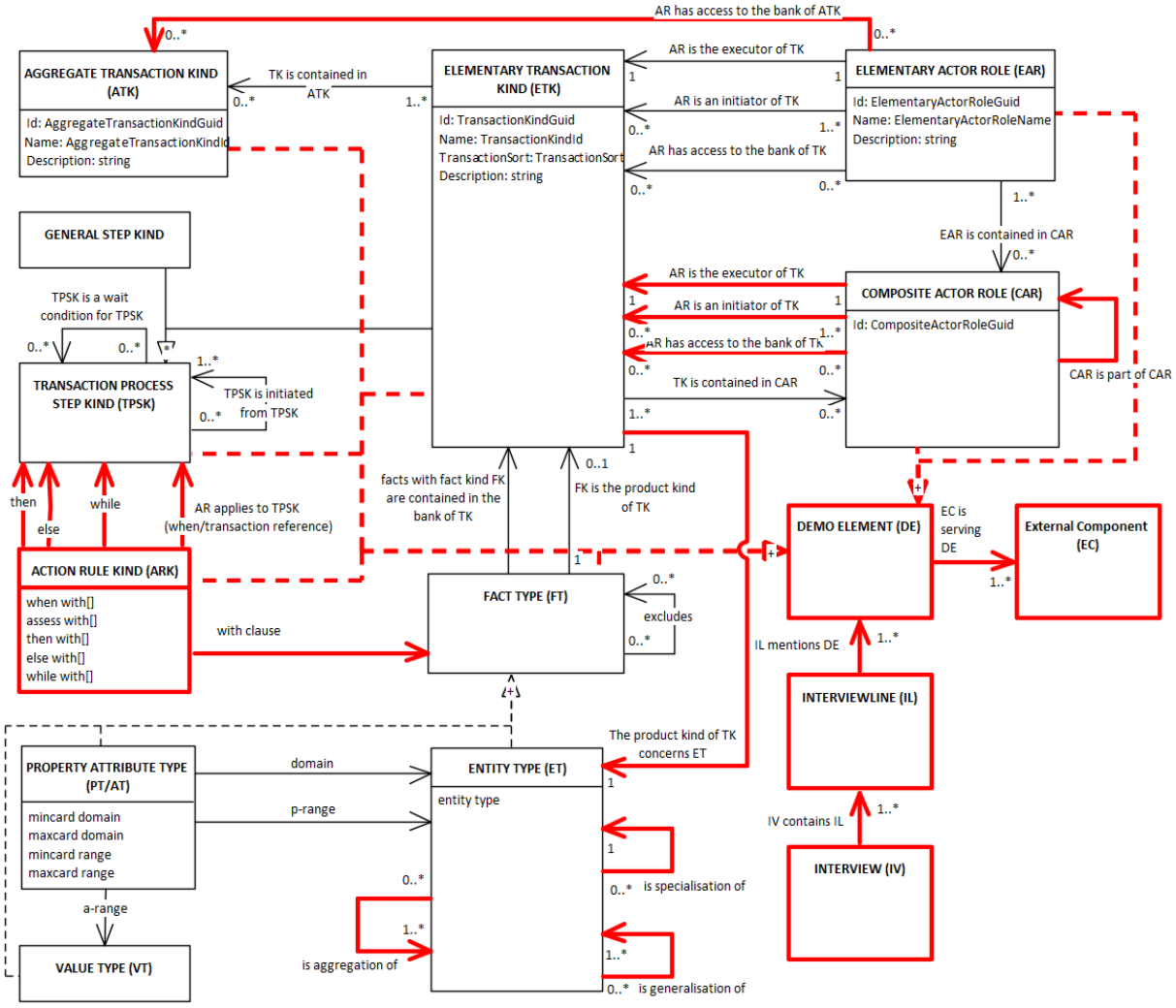


Figure 4.5: DEMOSL ontological-metamodel

The DEMO model is composed of four aspect models (CM, PM, FM, and AM). These models have their metamodel boundaries as has been visualised in fig. 4.6. Furthermore, these aspect models can also be visualised in diagrams and tables. All four aspect models are discussed in detail in sections 4.2 to 4.5.

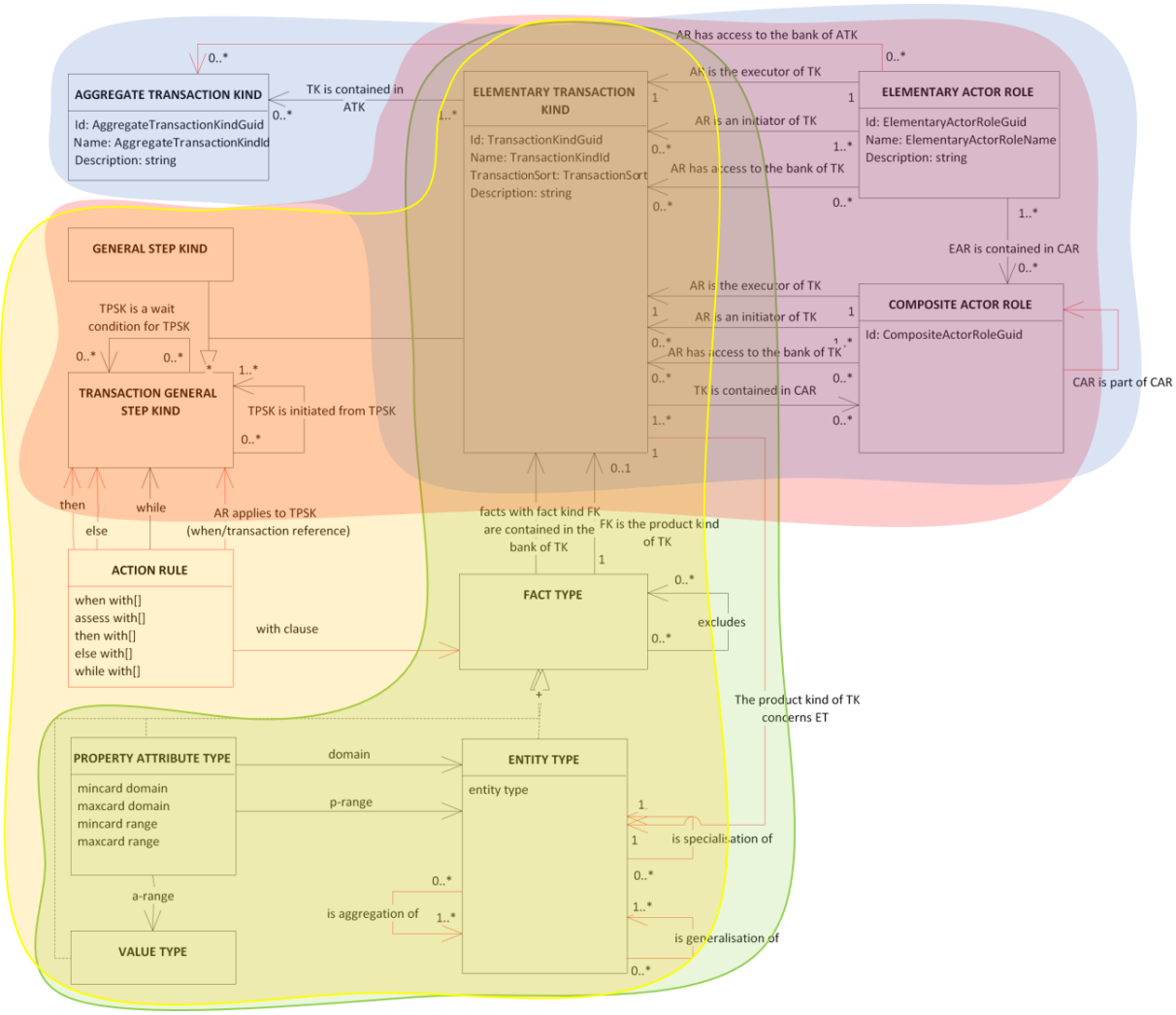


Figure 4.6: DEMO aspect models (Blue=Construction Model (CM), Red=Process Model (PM), Green=Fact Model (FM), and Yellow=Action Model (AM))

4.1.2 Verification-Metamodel

The verification rules that control the correctness of the DEMO model within the metamodel can be formulated in logical terms. Every entity type and property type in the metamodel can be represented as a predicate, with one or two parameters, respectively. In this thesis, we chose to underline the constant value of a written concept. Per entity type we have formulated all predicates of the metamodel. All equations used are targeted at a single organisation, regardless of the size of said organisation. The modularity of the DEMO methodology allows for the extension of an organisation within the same model. It is worth mentioning that if the specific property type has been explained in another part of this thesis, its predicate will not be described in detail in this chapter. The base equations for the complete ontological-metamodel are listed hereafter. The tag after the equation number is used to match the verification programming code. We will

start with the predicate of the DEMO model of organisation with elements x.
We use the mathematical symbols for 'and' $\wedge$, 'or' $\vee$ and 'xor' $\veebar$. Let x be a model element, then we should have:

$$\begin{aligned}\mathsf{DEMO}(x) \implies & \mathsf{ConstructionModel}(x) \\ \vee\ & \mathsf{ProcessModel}(x) \\ \vee\ & \mathsf{FactModel}(x) \\ \vee\ & \mathsf{ActionModel}(x)\end{aligned} \qquad \text{(E.1 ref) DEMO}$$

$$\begin{aligned}\mathsf{ConstructionModel}(x) \implies & \mathsf{ConstructionConcept}(x) \\ \vee\ & \mathsf{ConstructionRelation}(x)\end{aligned} \qquad \text{(E.2 ref) CM}$$

$$\begin{aligned}\mathsf{ProcessModel}(x) \implies & \mathsf{ProcessConcept}(x) \\ \vee\ & \mathsf{ProcessRelation}(x)\end{aligned} \qquad \text{(E.35 ref) PM}$$

$$\begin{aligned}\mathsf{FactModel}(x) \implies & \mathsf{FactConcept}(x) \\ \vee\ & \mathsf{FactRelation}(x)\end{aligned} \qquad \text{(E.49 ref) FM}$$

$$\begin{aligned}\mathsf{ActionModel}(x) \implies & \mathsf{ActionConcept}(x) \\ \vee\ & \mathsf{ActionRelation}(x)\end{aligned} \qquad \text{(E.63 ref) AM}$$

$$\begin{aligned}\mathsf{ActorRole}(x) \implies & \mathsf{ElementaryActorRole}(x) \\ \veebar\ & \mathsf{CompositeActorRole}(x)\end{aligned} \qquad \text{(E.5 ref) AR}$$

$$\begin{aligned}\mathsf{TransactionKind}(x) \implies & \mathsf{ElementaryTransactionKind}(x) \\ \veebar\ & \mathsf{AggregateTransactionKind}(x)\end{aligned} \qquad \text{(E.6 ref) TK}$$

The implementation of these verification rules in our automated tool is listed in appendix F.

4.1.3 Visualisation-Metamodel

Communicating a graphical representation of a DEMO aspect model is not trivial. There are different graphical representation options available, each with its advantages and disadvantages. One could choose to communicate the image format in BitMaP image file (BMP), Portable Network Graphics (PNG) or Scalable Vector Graphics (SVG) formats. These formats would give the model interpreter a visualised representation, but they would lack the underlying linked model information. We could also transform the graphical representation into a standard commercial format like PowerPoint XML Presentation (Microsoft) (PPTX) or Microsoft Visio XML Drawing (VSDX), but that could cause a vendor lock-in. Instead of these formats, we chose to only create the "essential" attributes of the kinds and types of the diagram. This point of view is new compared to the DEMOSL [1], which only describes visual properties and the exchange format [86] but does not address any visuals.
When we represent specific elements in a visualised graphical format, the location of that element on the diagram is an essential property. The location type attribute is available for purposes that might differ between software implementation. The size of the element, on the other hand, is used to visualise elements within a diagram. Finally, we defined a line as a visualised connection between two or more points. This line has a starting point, an endpoint and optionally several midpoints to create a path within the diagram.
The information shown in table 4.3 on page 128 provides all allowed Property Types between the objects of this metamodel, while in the next section we will discuss more specific visualisation details of the various visualisation aspect models.

```
sincerity: <no specific condition>
truth:
  for each Order Item in the contents of
      Order:
    the pizza kind of Order Item is
        available

       the amount of Order Item is
           producible
    the requested production time of
        Order is greater than Now plus 10
        minutes
```

```
truth:
  for each Order Item in {the contents of
      Order} {
    the availability of pizza kind of
        Order Item is equal to
        true;
    the [producebility of amount] of
        Order Item is equal to true;
    the requested production time of
        Order is greater than the [Now
        plus10 minutes] of Time;
  }
```

Figure 4.7: ARS grammar upgrade example

Due to the complexity reduction of the CM the resulting aspect model visualisation is not readable for stakeholders that expect a flow. This flow is only available when modellers have finished the analysis from CM through PM to AM. The DEMO specification has a language for the AM that is semi-formalised and we tried to create a formalised form for that model (see fig. 4.7). This specification, though formally more correct, is still not readable for many stakeholders. Visualising the final aspect model AM in a recognisable form should have positive effects on the stakeholders comprehension of the DEMO model because process visualisation on implementation level is widely spread.

When visualising an (aspect) model the visualisation is a representation of the model. On a meta-level one needs to define the elements that are visualised and the metamodel components of those elements that are used in this visualisation. The omission of this component visualisation definition makes the visualisation arbitrary. Defining these visualisations requires a mapping specification of every visualisation to the entity and attribute of the metamodel it is representing. These mappings also need translations from available values to required visualisations. For example, the transaction sort of a Elementary Transaction Kind (ETK) is defined as original, informational or documental and can be visualised as a red, green or blue diamond, respectively. While creating the models and tooling and trying to visualise the model for various stakeholders this omission of the mapping became apparent, whereas this never happens with the small examples from the educational material.
The mapping of the visualisation is the definition of viewpoints of DEMO modelling. At this moment just a few viewpoints have been defined in table 4.3, but more exist and can be defined for specific stakeholders. Viewpoints that cover multiple notations have to be defined to cover the representation combinations of DEMO and other notations (e.g. ArchiMate).
Each viewpoint needs, next to the definition and mapping, the semantic explanation of the information represented. Not only does every element and connection needs an explanation, but also the whole viewpoint needs an explanation, and even a definition. The current viewpoints (e.g. diagrams and tables) have neither definition nor explanation of what they represent. The OFD does show the entity types, attribute types, property types, and product types (the product type is another viewpoint of the ETK) but the representation definition is not clear. This OFD has been interpreted as the data model, the creation of data due to transaction acts, the relational model, the possible-state model and the authorisation model. Various interpretations of the OFD are possible and, therefore, it is likely that the OFD contains too much information. We have tried various diagrams with high level, detailed information, and all sorts of combinations to transfer the model to the stakeholder. Our experience tells us that the information density of each of the DEMO diagrams and tables is too much for the average stakeholder to grasp.

4.1.4 Exchange-Metamodel

As discussed in section 5.1.2, a number of tools can model (partial) DEMO models, while some of these even have a built-in engine to execute the actual models. It would be beneficial to be able to export and import DEMO models across tools because specific tools can have distinct benefits when used in a certain organisational environment.
In addition, given the complementarity between e.g. BPMN, ArchiMate, e3Value, and DEMO, it would also be beneficial to bridge the gap between the respective models. Experimental results regarding the potential benefits of this have been reported in e.g. [63, 64, 51, 87, 88]. Bridging the gap to other modelling tools requires DEMO modelling tools to be able to export (and import) DEMO models. To this end, an exchange-metamodel has been created that enables the exchange of both the actual model, including the different aspect models, as well as their visualisations. The exchange-metamodel enables trans-

lation and connection to various other languages and includes model extensions which allows to store models in other modelling languages such as ArchiMate and Business Process Model and Notation (BPMN). Due to its common availability in modern-day development environments, XSD is used as the exchange-metamodel base.
The SEA based modelling tool has been extended with the aforementioned export capability. This is done in order to verify the exchange-metamodel, and to experiment with the export of DEMO models towards a gamification application (as illustrated in fig. 6.6).
The concept of a data-exchange-metamodel is intertwined with the concept of a metamodel as explained in section 2.6. We will formalise the data-exchange-metamodel in terminology that a computer can understand. We have chosen to represent the exchange-metamodel in XSD, which is a commonly used technique. The alternative structure, XML Metadata Interchange (XMI), has no broad implementation and, as such, we rejected it as a potential candidate. The exchange-metamodel represents all entity types and property types of the ontological-metamodel. Furthermore, it is unambiguously readable for a computer. The proposal of a DEMO exchange format [89, 86], based on DEMO 2, is a good start for creating our exchange-metamodel.
Despite the current version of DEMO being version 4 (during writing of this thesis), the modelling itself has not changed and is still equivalent to DEMO 2. No attempt has been made to upgrade the proposed formats to DEMO 3 before. Above all, we have based our research on DEMOSL version 3.7, and, therefore, this version is already an upgrade of the existing format. DEMO 4 introduces a new element (transactor) but does not invalidate DEMO 2 or DEMO 3 models and, thus, the decision for choosing these older metamodels for representation of the exchange-metamodel are still valid. In future research this exchange-metamodel has to be adapted to the features of DEMO 4.
The existing paper [86] proposes an exchange format for the FM and the CM. The XSD for CM proposes the storage of the id, name, initiator(s), executor, and information link and result type of the transaction kind. This information is based on older DEMO specifications and, therefore, lacks essential information from version 3.7. We decided to build this XSD structure for every element type in the new DEMOSL proposal structure.
The first XSD structure is the "name" element. Just like any identifiable element, the whole DEMO model has to have a name. Next to the name, every component of the DEMO model has an internal id. We will not mention these attributes while using the exchange-metamodel, but we do model them.
The next XSD structure is a "Global Unique IDentfier (GUID)" element. A GUID identifies all types and kinds within the exchange-model and is a 128-bit number used to uniquely identify information in computer systems. In listing D.36 the reader can find the complete specification of this used type.
Another XSD structure contains "rule" elements. Rules restrict the naming of the elements used in e.g. a CM. One of the rules we would like to mention in this thesis is the transaction kind name rule. The name is built up of lower case words as specified in the DEMOSL [1]. The XSD notation enables enforcement of this rule. It is important to note that we allow empty names for practical use purposes during modelling. Name definitions are listed in appendix D.12
The full set of XSD specifications is already made available for evaluation purposes to

some research groups. The full specification with examples will be publicly available at the end of this research project. More information on the location of this full specification will be published on multiple sites [1].
A specific class of model exchanges needed in practice is the exchange between DEMO and 'Voorwaarden scheppen voor de Invoering van Standaardisatie ICT in de bouw' (VISI). As mentioned in section 2.1, VISI is an early descendant of DEMO which has now evolved into the ISO 29481-2:2012 (BIM-related) standard for the construction sector. The exchange model between DEMO and VISI has been defined in the standard ISO 29481-2:2012.

4.1.4.1 Data-Exchange-Metamodel

The core of the data-exchange-metamodel is the set of entity types, property types, and diagrams. The ontological-metamodel contains all concepts and references the model concept itself (DEMOModel) (fig. 4.8). Exchanging information of a DEMO model in such a way that the model can be reproduced at another machine makes it necessary that all these concepts are incorporated in the data-exchange-metamodel.

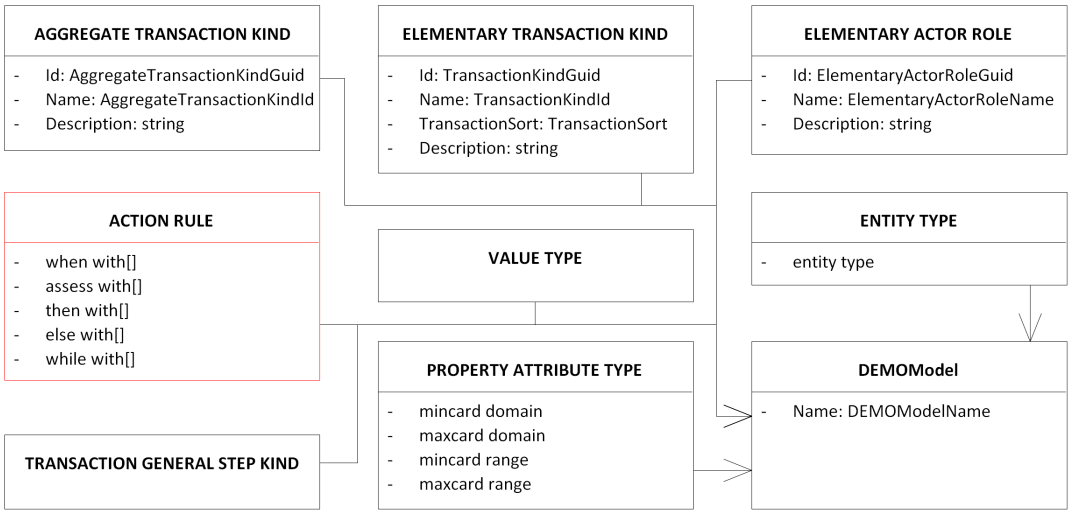


Figure 4.8: DEMOSL Core exchange-metamodel

The notation used for the data-exchange-metamodel is defined in the XSD schema specification (see listing D.1). The current site of the Enterprise Engineering institute (EEi) fully qualified domain name (FQDN) is the chosen namespace for the XSD.
The exchange model uses the element "DEMOmodel" in listing D.2 and the attributes (e.g. "DEMOModelName") in listing D.3 are used to identify the DEMO model.
The core model also has collections of every element. In appendix D.2 the reader can find the list of specifications for the core metamodel, whereas appendix D.4 lists the collection of diagrams.
Finally, the data-exchange-metamodel specifies a storage space for the verification results (see listing D.107 in appendix D.15 on page 94).

[1] http://demo.nl and http://teec2.nl

4.1.4.2 Visualisation Exchange-Metamodel

All elements that have been described in the previous section have a visualisation element in the exchange metamodel. The structure of the visualisation elements is similar to the data-exchange-metamodel structure. All visualisation elements reference the data elements in the same exchange model, and add the visualisation attributes for a specific diagram to the data element (e.g. size, location, way points).

4.2 Construction Metamodel

The CM is one of four aspect models that focuses on transaction kinds and actor roles.

4.2.1 Improvements in the CM

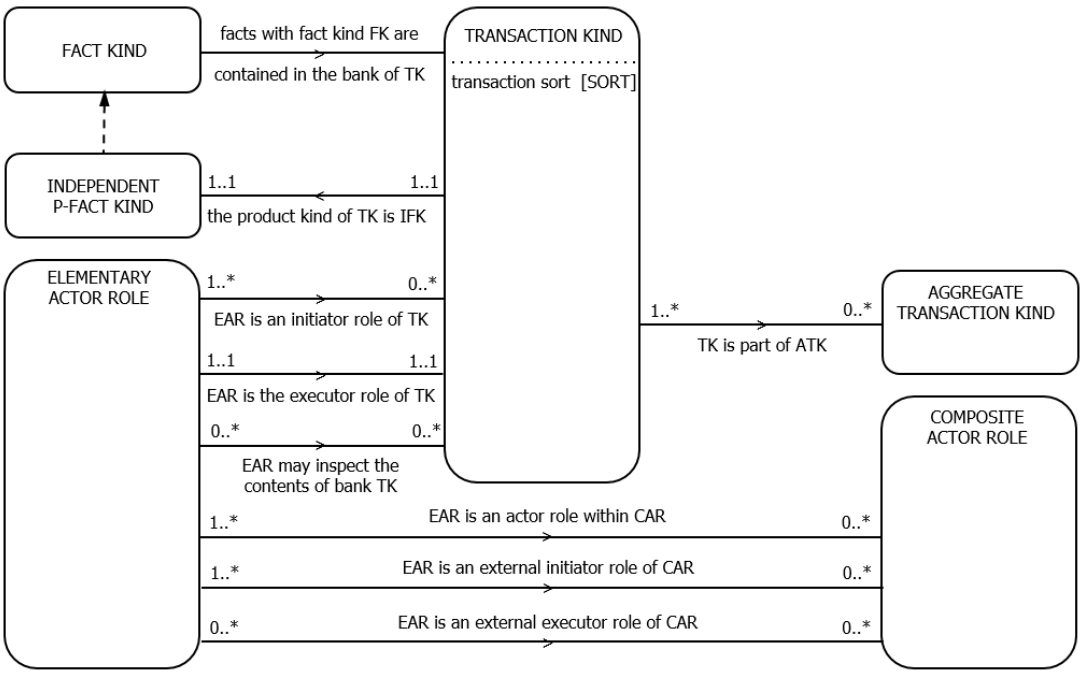


Figure 4.9: DEMOSL CM ontological-metamodel 3.7

From the validation, section 3.3.2, we know that we had to solve eight issues in the metamodel.

- CM.b the mandatory presence of a TK for an ATK;
- CM.c the mandatory presence of an EAR for a CAR.
- CM.d the missing access link from an EAR to an ATK;
- CM.e the lack of the SoI;
- CM.f the missing CAR to TK Property Types;
- CM.g the missing CAR hierarchy;
- CM.h the missing TK in CAR Property Types;

- CM.i the missing interstriction (access) between ATK, CAR and SoI;

The items will be referenced in all explanation of the formulas.

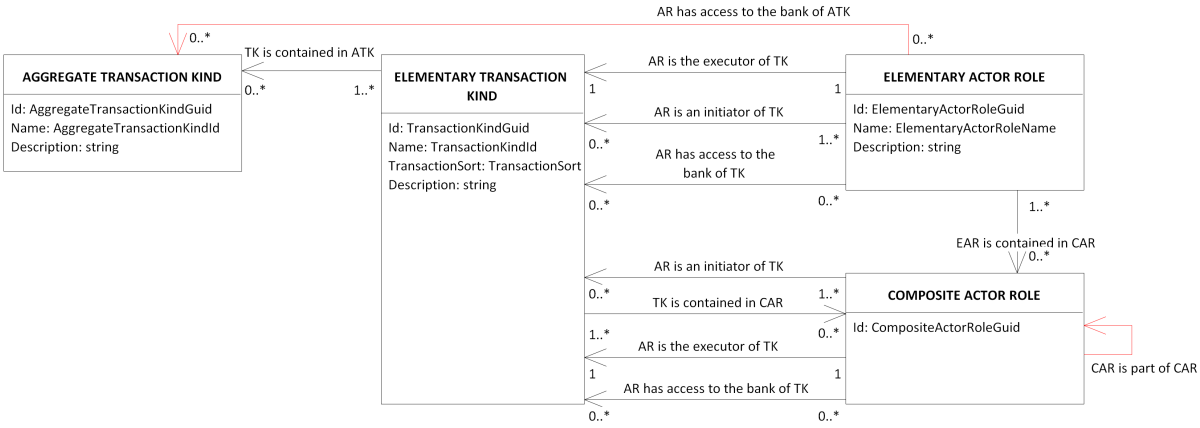


Figure 4.10: DEMOSL CM data metamodel proposal

$$\begin{aligned} \mathsf{ConstructionConcept}(x) \implies & \mathsf{TransactionKind}(x) \\ \vee\ & \mathsf{ActorRole}(x) \end{aligned} \qquad \text{(E.3 ref)}\ \text{CMcs}$$

Source and target kinds and types are referenced by the s and t respectively.

$$\begin{aligned} \mathsf{ConstructionRelation}(\langle s,t\rangle) \implies & \mathsf{executor}(s,t) \\ \vee\ & \mathsf{initiator}(s,t) \\ \vee\ & \mathsf{access}(s,t) \\ \vee\ & \mathsf{TKcontainedinATK}(s,t) \\ \vee\ & \mathsf{TKcontainedinCAR}(s,t) \\ \vee\ & \mathsf{ARaccessATK}(s,t) \\ \vee\ & \mathsf{EARcontainedinCAR}(s,t) \\ \vee\ & \mathsf{CARcontainedinCAR}(s,t) \end{aligned} \qquad \text{(E.4 ref)}\ \text{CMrel}$$

4.2.2 Organisation Construction Diagram

The OCD is a diagram that expresses a part of the CM. More specifically, it contains the TK, ATK, EAR and CAR concepts. The visualised property types are the initiator, executor, and access property types (see section 4.7).
The representation of the OCD is used for the communication of the CM. The OCD allows TK, ATK, EAR, CAR as valid elements and initiator, executor and access (interstriction) as valid property types.

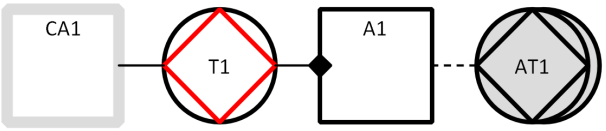


Figure 4.11: DEMOSL CM OCD visualisation

The OCD can be represented using logic. The main equations are written in the eqs. E.27 to E.30.
The exchange model for an OCD has elements for the TK, ATK, EAR, CAR, and connections (property types). Together with an id, name, and formulation these elements and connections make up the complete OCD. The XSD code can be found in listing D.108.

4.2.3 Transaction Product Table

The visualisation of the transactions and their products is a table with four columns: the transaction kind identification, the transaction kind name, the product kind identification and the product kind formulation. The first two columns are filled by transaction kind attributes Identification and Name. The last two columns are the properties Identification and Name of the Independent Fact Kind. These two elements have a one-to-one relation and always exist. The main equations are written in the eqs. E.7, E.31, E.32 and E.56.

4.2.4 Bank Content Table

The BCT is the list of information that is stored in the bank. It is a collection of entities and attributes (facts). The main equations are written in the eqs. E.33, E.34 and E.59.

4.2.5 Elementary Transaction Kind

A TK represents the abstraction of a transaction as defined in the PSI-theory. An summary of this theory has been presented in section 2.1.1. The TK concept has a relation to the concept of ATK where it can be contained to form bigger information banks. It can also be contained in a CAR where it can hide to be part of an underlying scope of interest. Compared to DEMOSL 3.7 [1] we added the contained-in property type between a TK and the CAR elements. This property type has been formulated in eq. E.10 *'TK-contained-in-CAR'*.

4.2.5.1 Ontological-Metamodel

The ontological-metamodel in fig. 4.12 shows the TransactionSort attribute and the description. The last attribute was requested by the focus group to be able to exchange meta-information about the TK.

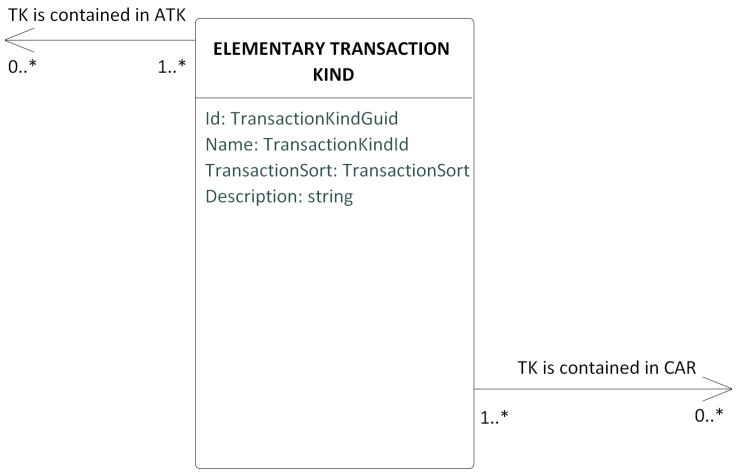


Figure 4.12: TK ontological model

4.2.5.2 Verification Rules

The base of the verification rules of an elementary transaction kind is that the TK is part of the collection of elementary transaction kinds. Equation E.8 (page 107) is the consequence of the eq. E.6 (page 107) that states that a collection of transaction kinds is the union of the elementary and aggregate transactions.
According to the PSI-theory the executor and the initiator of a transaction kind should be an actor role. This can be an elementary actor role or a CAR as outlined in eqs. E.7 and E.8.
When the executor of a TK is an EAR, and this EAR is part of a CAR, the CAR is also an executor of the TK. This equation is derived in section 4.2.7.2.
The ontological-metamodel property types can be shown in a logical way. The property types in the ontological-metamodel denote that a TK can be contained in both an ATK and a CAR. We can conclude from the ontological-metamodel that eq. E.9 and eq. E.10 should hold. This last equation solves issue CM.h.
When a TK is contained in a CAR, the initiator (eq. E.8) and executor (eq. E.7) of that TK must also be part of the CAR as in eq. E.11 and eq. E.12. Only then can we assure that the initiation and execution of other transactions are well defined.
The example in fig. 4.13 shows that CA4, where T2 and T3 are part of CA4, does not comply with eq. E.11 and E.12 because it is missing the involvement of A1 and A3.

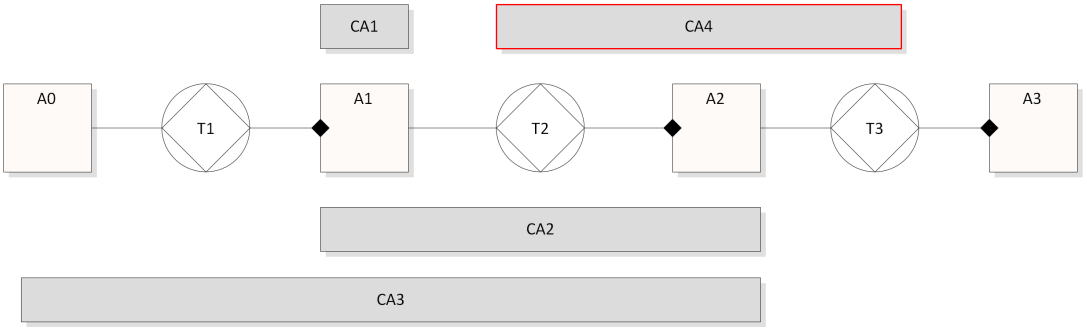


Figure 4.13: TK contained in CAR example

Next, the number of TKs should be equal to the number of executing Action Research (AR), if and only if the executing AR is the lowest (or elementary) executor of that transaction.

$$\mathsf{ElementaryTransactionKind}(t) \implies \exists_a \left[\mathsf{executor}(t, a)\right] \qquad \text{(E.13 ref)}\ _{\text{TKexistAR}}$$

$$\mathsf{ElementaryTransactionKind}(t) \implies \exists_a \left[\mathsf{initiator}(t, a)\right] \qquad \text{(E.14 ref)}\ _{\text{TKexistARi}}$$

$$\begin{aligned} \mathsf{ElementaryTransactionKind}(x) \ &\wedge \\ \mathsf{ElementaryActorRole}(y) \ &\wedge \\ \mathsf{executor}(x, y) \ &\wedge \\ \mathsf{initiator}(y, x) \implies \mathsf{SelfInitiator}(y) & \end{aligned} \qquad \text{(E.15 ref)}\ _{\text{ARself}}$$

4.2.5.3 Exchange-Metamodel

A transaction in DEMOSL 3.7 has the properties of a name, transaction sort, and an identification. We discussed in the focus group and agreed to implement the ID of the exchange data as an attribute. The code is listed in listing D.27 and given as example in this section.

```
<xs:complexType name="TransactionKind">
  <xs:sequence>
    <xs:element name="Identification" type="TransactionKindId"></xs:element>
    <xs:element name="Name" type="TransactionKindName"></xs:element>
    <xs:element name="TransactionSort" type="TransactionSort" default="
       unknown"></xs:element>
```

```
  </xs:sequence>
  <xs:attribute name="Id" type="TransactionKindGuid" use="required"/>
</xs:complexType>
```

Listing 4.1: Transaction kind

This exchange rule set described in XSD can sufficiently restrict field content for each TK.

4.2.6 Aggregate Transaction Kind

ATKs may be designed with or without contained TKs. In the organisation the ATK is outside the SoI, therefore, we don't know about the contained TKs.

4.2.6.1 Ontological-Metamodel

The ATK has the same attributes as the TK, except for the TransactionSort (fig. 4.14).

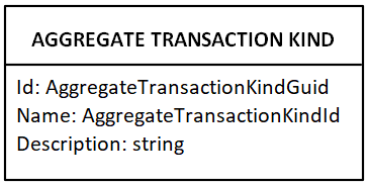


Figure 4.14: DEMOSL ATK metamodel

4.2.6.2 Verification Rules

As can be seen in fig. 4.14, no property types are coming from the ATK. Therefore, no collection equations have been formulated for the ATK.

$$\mathsf{ARaccessATK}(x,y) \implies \mathsf{ActorRole}(x) \wedge \mathsf{AggregateTransactionKind}(y) \quad \text{(E.16 ref)}\ {}_{\text{ARaccATK}}$$

$$\mathsf{FTcontainedinTK}(x,y) \implies \mathsf{FactType}(x) \wedge \mathsf{TransactionKind}(y) \quad \text{(E.59 ref)}\ {}_{\text{FTinTK}}$$

4.2.6.3 Exchange Metamodel

In the exchange model, the ATK is modelled explicitly. Apart from the TransactionSort element, that is left out, this exchange model is equivalent to the TK exchange model as shown in listing D.28.

4.2.7 Elementary Actor Role

4.2.7.1 Ontological-Metamodel

The EAR data-metamodel has five property types to other elements (fig. 4.15).

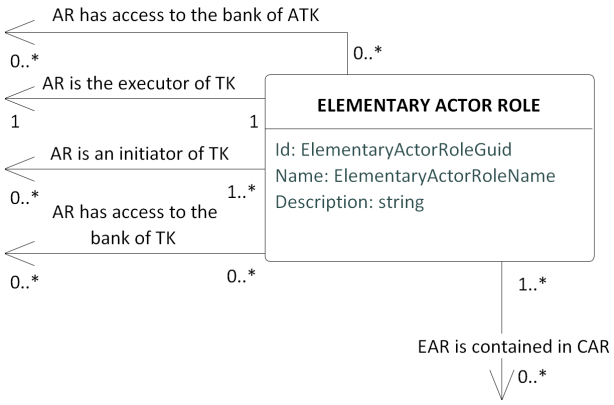


Figure 4.15: DEMOSL EAR metamodel

4.2.7.2 Verification Rules

Verification rules for the EAR are listed in eqs. E.7, E.8 and E.18. Equation E.19 makes sure a TK can only be executed by a single EAR.

$$\begin{aligned} \mathsf{executor}(x,y) \ &\wedge \\ \mathsf{executor}(x,z) \ &\wedge \\ \mathsf{ElementaryActorRole}(y) \ &\wedge \\ \mathsf{ElementaryActorRole}(z) \ &\implies y = z \end{aligned} \tag{E.19 ref EARexs}$$

Equation E.20 solves the issues CM.b and CM.d of sections 3.3.2 and 4.2.1 by adding the access directly to an TK and adding the possibility to create an access property type to an EAR. Next, this equation solves issue CM.i, CM.f, and CM.c.

$$\mathsf{ARaccessETK}(x,y) \implies \mathsf{ActorRole}(x) \wedge \mathsf{ElementaryTransactionKind}(y) \tag{E.20 ref ARaccETK}$$

The executor of a transaction kind should be an actor role. This can be an elementary actor role or a CAR. When the executor of a TK is an EAR, and this EAR is part of a CAR, the CAR is consequently also an executor of the TK. We use the formulation of CAR and EAR property types, i.e. eq. E.22. As seen from the execution viewpoint of the transaction kind we get eq. E.21.

$$\begin{aligned} &\mathsf{ElementaryTransactionKind}(x) \ \wedge \\ &\mathsf{ElementaryActorRole}(y) \ \wedge \\ &\mathsf{CompositeActorRole}(z) \ \wedge \\ &\mathsf{executor}(x,y) \ \wedge \\ &\mathsf{executor}(x,z) \implies \mathsf{EARcontainedinCAR}(y,z) \end{aligned} \qquad \text{(E.21 ref)}_{\text{TKme}}$$

The graphical representation of eq. E.21 is in fig. 4.16.

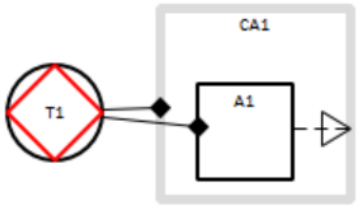


Figure 4.16: DEMOSL EAR in CAR as executor

4.2.7.3 Exchange Metamodel

The listing listing D.29 shows the EAR specification.

4.2.8 Composite Actor Role

The CAR is the embodiment of multiple actor roles at once. This concept allows for modelling the unknown actor role or the 'don't want to know' actor role.
In the DEMO methodology the concept of SoI is used to restrain the modelling effort to a selected portion of the organisation. Absent from the DEMOSL metamodel [1], is the SoI itself. To be able to model this concept it could be added, but closer examination of the concept reveals something more interesting.
We argue that *the concept of SoI is equivalent to the concept of CAR within the modelling of CM.* When starting to model an organisation, the first actor inside the SoI is a composite actor role. The methodology describes that this composite actor role stays in place until one is able to retrieve the information to redesign the internal actor roles of this composite actor role into a white-box model. The CAR does not vanish. Instead, the CAR becomes functionally equal to the SoI as can be seen in fig. 4.18 and 4.19. Therefore, the only difference between an SoI and a CAR is its appearance in the diagram. The boundary transactions of the SoI are the interacting transactions in the diagram.
One could still argue that a SoI could be smaller than all transactions that interact with the CAR. This can be illustrated with an extra transaction to CA0 in the construction model that is not shown in 4.18. The question that remains is whether the CAR, used as SoI is the same in the diagram as in the model. In practice the SoI never exceeds the diagram boundaries and therefore, de-facto, the CAR can be used for the same purpose. When using the CAR as SoI some nice features for the hierarchy of actor roles appear.

This being said, combined with the ability to nest CARs it becomes clear that an EAR within multiple CAR is not modelled as a construction. It is possible to model this hierarchy as a functional structure, though this is not ontological. Therefore eq. E.26 has been added.

4.2.8.1 Ontological-Metamodel

In the CM, the CAR and the EAR are modelled as separate entity types. By modelling the CAR (fig. 4.17) as a specialisation of an EAR, the property types of the elementary actor role are also available for the composite actor role. This change allows us to model the composite actor role 'customer' in the pizza case [25, fig 16.8] where the customer is the initiator and executor of a TK without any elementary actor roles present.

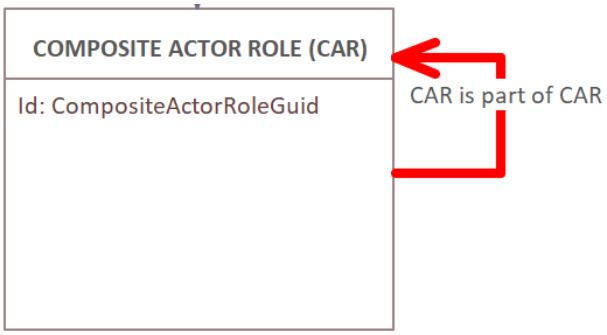


Figure 4.17: DEMOSL CAR metamodel

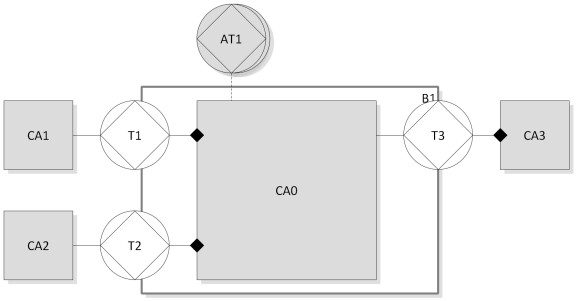


Figure 4.18: CM modelling step 1

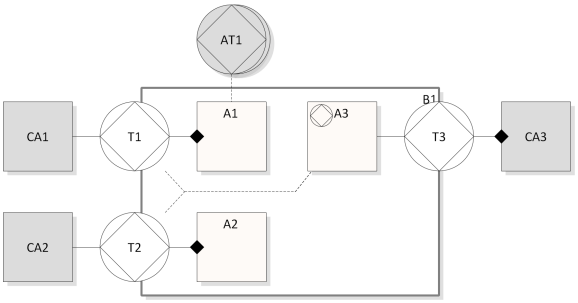


Figure 4.19: CM modelling step 2

4.2.8.2 Verification Rules

The *CARispartofCAR* in fig. 4.17 property type describes that the CAR contains another level of CAR. This means that if a CAR is described in more detail, it is connected to the CAR it belongs to. We have formulated these properties in eqs. E.22 to E.25.
When an EAR can belong to multiple CARs, the CARs have been chosen from a functional perspective. In DEMO this perspective is not preferred. Often the construction CAR is confused with the functional concept of a department.

$$\mathsf{EARcontainedinCAR}(x, y) \ \wedge \\ \mathsf{EARcontainedinCAR}(x, z) \implies y = z \qquad \text{(E.26 ref)}_{\text{EARxinCAR}}$$

For instance, when modelling an organisation, the functional components like departments are needed for stakeholders to understand the structures of the organisation. Given DEMO's focus on the ontological essence of organisations, it does not provide functional concepts like departments. It does, however, provide the constructional concept of Composite Actor Role (CAR). CARs are now allowed to be used to represent functional concepts such as departments, enabling modellers to model departments (or other organisational units) as organisations within organisations. This hierarchical relation has been added to the metamodel.

4.2.8.3 Exchange-Metamodel

The exchange model of the CAR is equivalent to the EAR. The type references have been renamed to match the composite intention. The listing listing D.30 shows the EAR specification.

4.3 Process Metamodel

4.3.1 Improvements in the PM

In the PSD the initiation of the first request is done using the property type 'is initiated from'. This would be sufficient if a step is always initiated from the same step. When, instead of the promise-step, another step is used for initiation this cannot be modelled. Therefore, the property type has to be remodelled as a self-reference in TPSK 'TPSK is initiated from TPSK'.

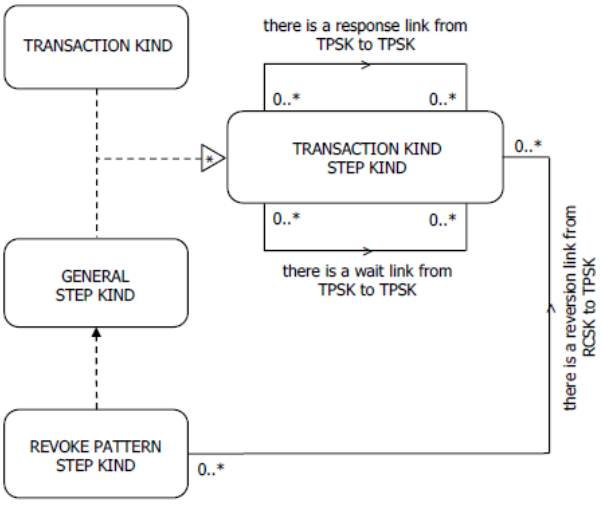


Figure 4.20: DEMOSL PM metamodel 3.7

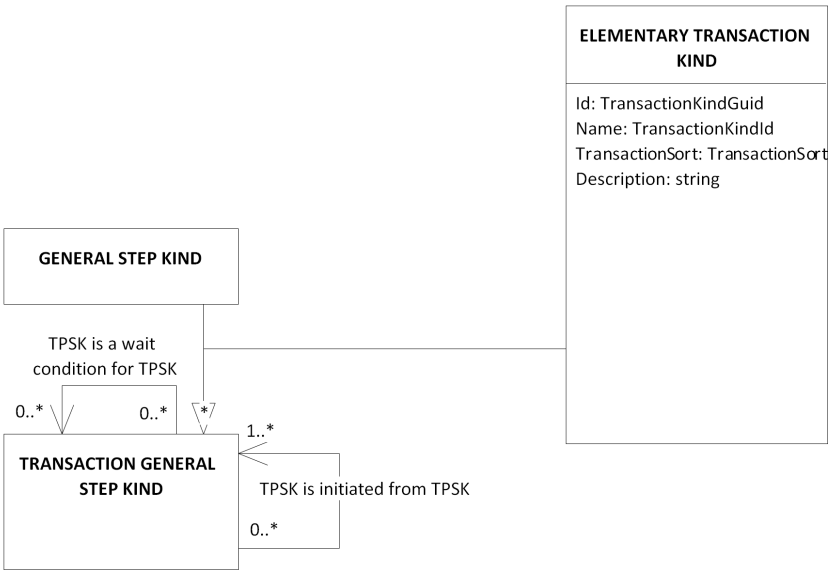


Figure 4.21: DEMOSL PM metamodel proposal

4.3.2 Process Structure Diagram

The PSD is a diagram that contains elements from the PM. More specifically, it contains the TK, EAR, and CAR. The visualised property types are the call-link and the wait-link property types (see section 4.7)

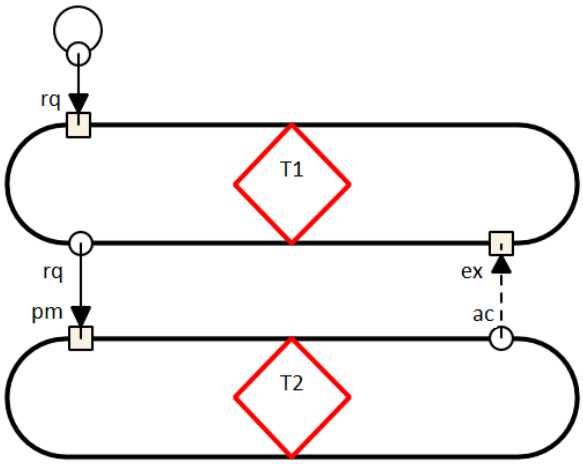


Figure 4.22: DEMOSL PSD visualisation proposal

The PSD can be represented using first-order logic. The main equations are written in eqs. E.38 to E.40.
The exchange model for a PSD has elements for the TK, initiator, EAR, CAR, and Connections. Together with an id, name, and formulation these elements are sufficient to describe the complete PSD. The XSD code can be found in listing D.109.

4.3.3 Transaction Process Diagram

The TPD is a diagram that contains elements from the PM. More specifically it contains the TK and TPSK. The visualised property types are the process order, call link and the wait link (see 4.7)

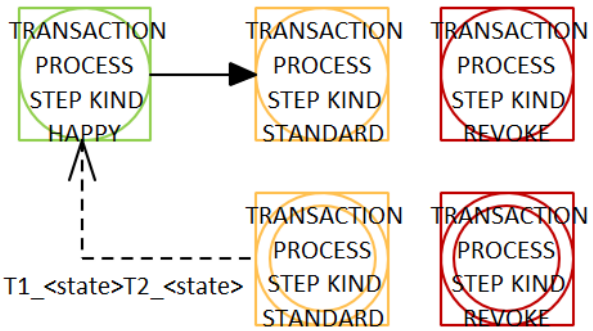


Figure 4.23: DEMOSL TPD visualisation proposal

The TPD equations are shown in eqs. E.41 and E.42. The exchange model for a PSD has elements for the TK, TPSK, and Connections. Together with an id, name, and formulation these elements are sufficient to describe the complete TPD. The XSD code can be found in listing D.110.

4.3.4 Elementary Transaction Kind

The ontological-metamodel and exchange model have already been presented in section 4.2.5.1 and 4.2.5.3, respectively. To apply the TK in the PM environment only extra verification rules need to be introduced.
A transaction, combined with the general step kind, will lead to a transaction process step kind. This equation will be explained in the TPSK in section 4.3.6.2.

4.3.5 General Step Kind

The GSK is a concept to enumerate all possible steps in the complete transaction pattern. This is mainly used to create the cross join between the transaction kinds and the possible transaction steps.

4.3.5.1 Ontological-Metamodel

The ontological-metamodel is a single entity representing all instances of acts and facts in the C-world and the P-world.

Figure 4.24: DEMOSL GSK metamodel

4.3.5.2 Verification Rules

We define a general step kind of a TPSK as the TPSK with the signature of the GSK. A subset of all GSKs is the set of request step kinds as shown in eq. E.44. A TPSK can only be of a single GSK. This is formulated in formulaE.46.

4.3.5.3 Exchange-metamodel

General step kind is enumerated in appendix D.11, listing D.64. It consists of all steps, the act, and the fact. These steps start with initial, request, requested, etc. The list also contains all revoke acts and facts, and the formalised notation of the words.

4.3.6 Transaction Kind Step Kind

A TPSK is the smallest ontological process concept. It represents the steps taken in every TK.

4.3.6.1 Ontological-Metamodel

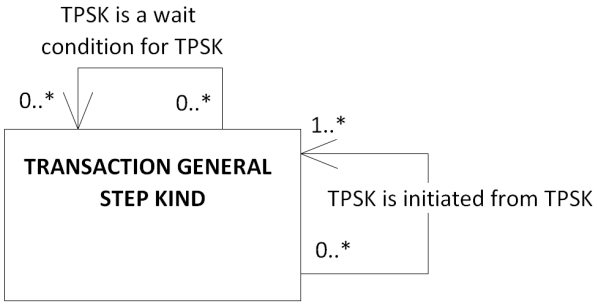


Figure 4.25: DEMOSL TPSK metamodel

4.3.6.2 Verification Rules

For every transaction process step kind there is a corresponding transaction kind as shown in eq. E.47. Furthermore, every TPSK request (all five) can be initiated from any TPSK from another or the same transaction as shown in eq. E.48.

4.3.6.3 Exchange-Metamodel

The listing listing D.31 shows the TPSK specification.

4.4 Fact Metamodel

The fact metamodel is about the data perspective of the DEMO models. There is a vast amount of literature available on the topic of data modelling. Therefore, we are not going to dive into the basics of data modelling and will focus on data perspective of DEMO methodology instead. We formulate relations (or property types):

$$\mathsf{relation}(x,y) \implies \mathsf{pdomain}(x) \wedge \mathsf{prange}(y) \qquad \text{(E.58 ref)}\ \text{ATrel}$$

This rule formulates an attribute:

$$\mathsf{attribute}(x,y) \implies \mathsf{adomain}(x) \wedge \mathsf{arange}(y) \qquad \text{(E.57 ref)}\ \text{AT}$$

And the relation between transaction kinds and the created data:

$$\mathsf{event}(x) \implies \mathsf{baseEntity}(x) \qquad \text{(E.61 ref)}\ \text{Event}$$

$$\exists_x \left[\mathsf{baseEntity}(x) \implies \mathsf{event}(x)\right] \qquad \text{(E.60 ref)}\ \text{BE}$$

Relations (or property types) are unique in the model:

$$\begin{aligned} \mathsf{property1}(x,y) &\implies \neg\mathsf{property2}(x,y) \\ \mathsf{property2}(x,y) &\implies \neg\mathsf{property1}(x,y) \end{aligned} \qquad \text{(E.62 ref)}\ \text{PT}$$

In the FM we have an event type entity type. This entity type is intended for modelling the transaction kind product in the FM. When we combine all partial ontological-metamodels, the event type entity type is no longer valuable. The transaction kind represents the same concept, the transaction kind product. Therefore, the event type entity type can be replaced by a property type between TK and primal entity type.

4.4.1 Improvements in the FM

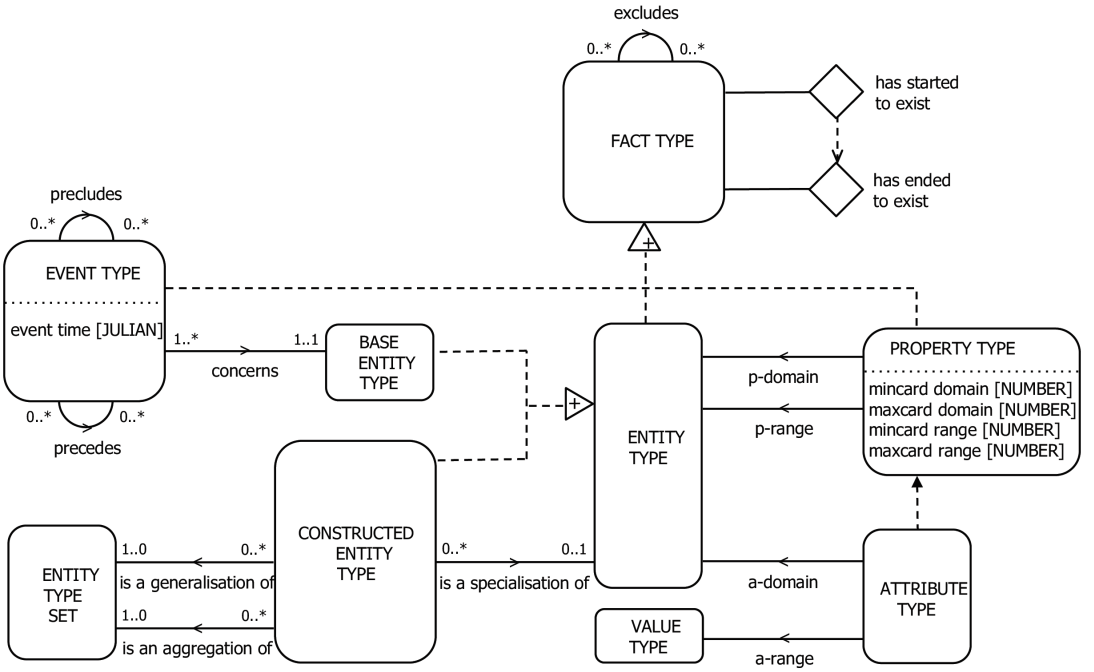


Figure 4.26: DEMOSL FM metamodel 3.7

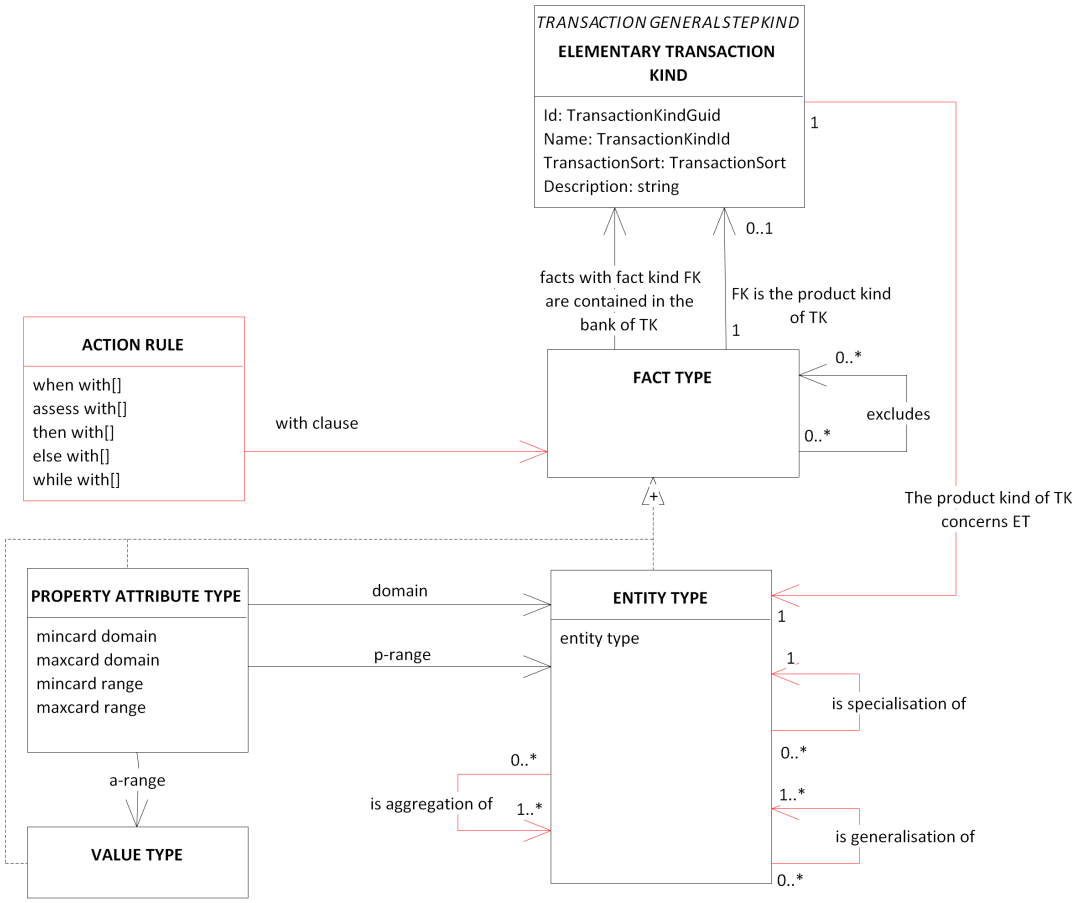


Figure 4.27: DEMOSL FM data metamodel proposal

The original ontological verification rules have some concepts that have made it directly to the ontological-metamodel of the fact aspect model, including Fact Type and Entity Type concepts. Other concepts will be removed or reduced because of the following reasoning. First, we have removed the Constructed Entity Type as it is a concept that allows reasoning about the specialised entities derived from an Entity Type. In the data model proposal this results in the Entity Type, like the generalisation property type between Constructed Entity Type and Entity Type suggested in the DEMOSL 3.7 FM metamodel. The same holds for the generalisation and aggregation property types of this constructed entity type. Both property types end in the Entity Type Set which, in turn, is an Entity Type concept on its own. This leaves the three property types as self-property types on the Entity Type. Next, we have also omitted the Event Type concept which is a data perception of the process events of transaction kinds and TPSK. Events are related to C-acts and C-facts and to a concern concept in the P-world. Therefore, the concerns property type is created between the Elementary Transaction Kind and the Fact Type concept as shown in eqs. E.50 and E.51.

4.4.2 Object Fact Diagram

The OFD is a diagram that contains elements from the FM. More specifically it contains the IFK, ET, and AT. The visualised property types are the reference, generalisation, specialisation, aggregation, and the concerns property types (see section 4.7)
The main underlying equations are written in the eqs. E.52 to E.55.
The OFD is listed in appendix listing D.111.

4.4.3 Fact Type or Independent Fact Kind

The Independent P-Fact Kind (IFK) is the corresponding fact that concerns the transaction. It has a product formulation, product kind, and a 1:1 relation with the transaction. The information about an IFK is stored in the FACT TYPE entity. Attributes are not added to this entity.

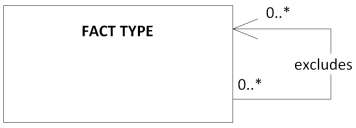


Figure 4.28: DEMOSL IFK metamodel

$$\mathsf{concern}(x, y) \implies \mathsf{ProductKind}(x) \wedge \mathsf{ElementaryTransactionKind}(y) \quad \text{(E.56 ref)}\ _{\text{PKconTK}}$$

For every transaction(x) there is a corresponding product(y). This product has a formulation. This formulation contains at least the base entity type name and optional other referenced entity type names.
The IFK is listed in appendix listing D.32.

4.4.4 Entity Type

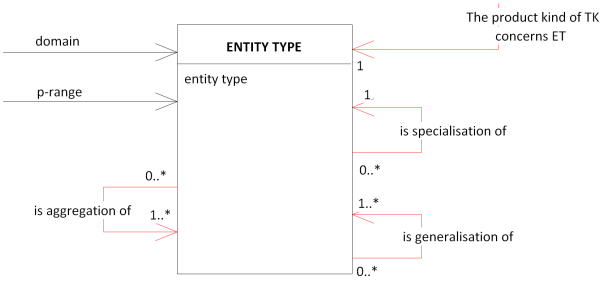


Figure 4.29: DEMOSL ET metamodel

The ET is listed in appendix listing D.33.

4.5 Action Metamodel

The AM is the aspect model with the largest amount of details about the enterprise. These details are largely based on the CM, PM, and the FM. Therefore, references to these aspect models should be used in order to keep the AM consistent with the other aspect models. Traditionally the ARS is specified in a natural language and is only partially covered by a incomplete grammar. This allows for freedom of expression while being readable for humans. Though the advantage of the readability is large, the disadvantage of the informal language makes it difficult, and at this moment even impossible, to do an automated model verification and validation. Where the other three aspect models have always been described in a graphical way, the AM has been described in a textual way with a semi-formal grammar construction. This makes communication towards different users of these models unnecessarily complex. All aspect models must have a textual and graphical representation of at least the essential features of the model to serve both stakeholder groups with their visual preferences.

4.5.1 Improvements in the AM

We added a graphical representation to the AM and improved the grammar of the ARS. All grammar references to other elements of the DEMO model have been connected in the action model.
The AM is the least developed and least used aspect model of DEMO. For automation we need this model to represent all the details about business rules and their property types to facts and processes.
The AM has no graphical representation in DEMO 3.7. One might argue that the distinction between the event, assess, and result parts is actual graphical representation of AM due to the fact that in educational slides these sections of the grammar are often visualised with a background colour. Nevertheless, this representation is not different from the colouring of a grammar in a modern integrated developing environment.
The existing AM has four essential features.

1. ET linked to a TK; it needs parameters (with-clause) to verify that all essential information is present to perform the step.
2. ARK linked to another step; it has to wait for other TPSKs to finish
3. Inner conditions defined in the ARK have to be checked.
4. Based on these conditions the linked follow-up actions have to be triggered.

Except for the feature number three, the features have a connection to other parts of the DEMO model. Connections can be visualised as a line, like it has been done in the other three aspect models.
So, let us define the connections for the AM.

- With
  The with-clause (see section 4.5.3.12) connects several attributes of entities (from the FM) to either the agendum-clause, while clause, or action clause. These connections will be represented with a solid line with a winged arrow head. The with connection can connect to the when, while, and action clauses.

- While
  The while clause (see section 4.5.3.11)connects a different transaction step (from the PM). This connection will be represented with a dotted line with a solid arrow, just like the wait link in the PSD.
- Action
  The action clause connects a transaction step with another step. This is the same connection as a call link in the PSD.

4.5.2 Action Rule Diagram

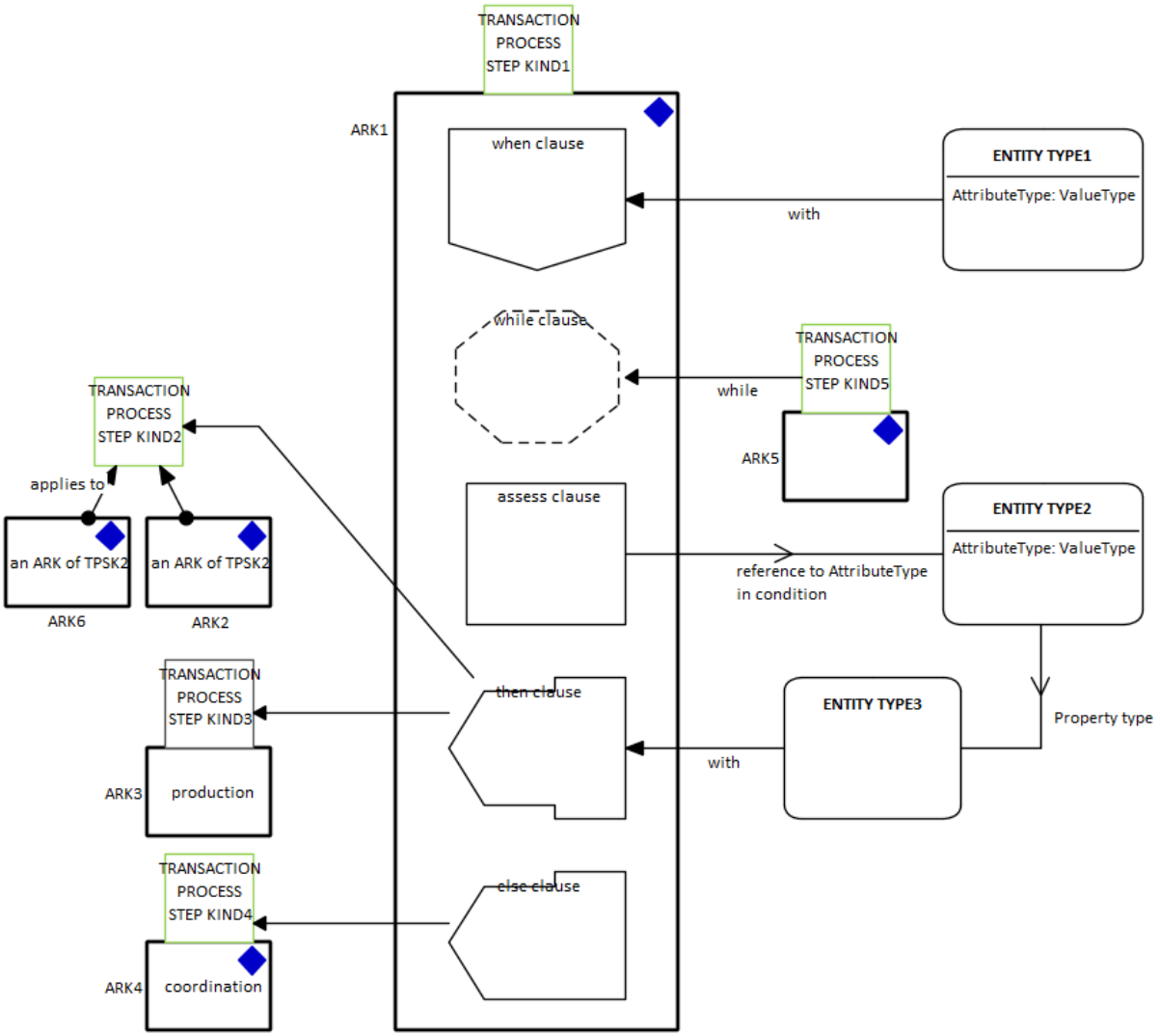


Figure 4.30: ARD representation example

Figure 4.30 gives an impression of the visualisation of this Action Rules Diagram (ARD). While not all arrows have been drawn to the right specifications, this early representation does give a instantiated view on the metamodel.

When we follow the example of fig. 4.30, and start at the TPSK1 placed on the top of the diagram, we can see the process unfolding. For the TPSK1 the Action Rules Kind (ARK) ARK1 is the action rule that needs to be evaluated. In the when clause the required information can be expressed with entity and attribute types. In the following step, the while clause, the process can wait on the completion of the TPSK 5. When these steps have been completed the assess clause evaluates whether the conditions, that reference the information in ET2, are true. If that is the case, next then clause triggers TPSK2 and as a result ARKs 2 and 6 are evaluated.
The Action Rules Diagram (ARD) is a diagram that visually represents a single action rule, like the ARSs represents the same information in a textual way. The ARD and ARS are representations of the same information. This representation of the AM has following elements for visualisation purposes: TPSK, ET, AT, and Grammar components when, while, assess, if then and else.
The ARD can be represented using first order logic. The main equations are:

$$\mathsf{ardDiagram}(d) \implies \mathsf{DiagramType}(d) \qquad \text{(E.66 ref)}\ _{\text{ARDd}}$$

$$\begin{aligned} \mathsf{ardDiagramElement}(x, d) \implies & \mathsf{ActionConcept}(x) \\ & \wedge\ \mathsf{ardDiagram}(d) \end{aligned} \qquad \text{(E.67 ref)}\ _{\text{ARDi}}$$

$$\begin{aligned} \mathsf{ardDiagramRelation}(r, d) \implies & \mathsf{ActionRelation}(r) \\ & \wedge\ \mathsf{ardDiagram}(d) \end{aligned} \qquad \text{(E.68 ref)}\ _{\text{ARDr}}$$

$$\begin{aligned} \mathsf{ardDiagramRelation}(\langle s, t \rangle) \implies & \mathsf{when}(s, t) \\ & \vee\ \mathsf{while}(s, t) \\ & \vee\ \mathsf{with}(s, t) \\ & \vee\ \mathsf{trigger}(s, t) \end{aligned} \qquad \text{(E.69 ref)}\ _{\text{ARDrel}}$$

The exchange model has elements for the TPSK and the connections. Other elements are yet to be added. The XSD code can be found in listing D.112.

4.5.3 Action Rule Specification

The information in the existing AM is not detailed enough to verify a model. After having analysed the existing, published action specifications we have created a grammar that closely represents the existing examples and matches the wishes of the expert group that

helped to create the exchange model. The verbalisation used in the DEMOSL grammar of the AM can also be specified in property types to the other aspect models. These property types are described in the ontological-metamodel in section 4.5.3.1, whereas the logical representation of these rules has been explained in section 4.1.2. Contrary to the information presented in the previous parts, this section will describe all relevant rules one by one referencing different formats. First, the grammar notation for the partial rule will be presented, followed by the exchange model notation of the same part. Where the rules are trivial only a reference is made to the complete set of rules for the DEMOSL proposal (appendix E), to the grammar (appendix C) and to the Exchange metamodel (appendix D)

Original grammar		Listing	Page	Original grammar		Listing	Page
Slide	Line			Slide	Line		
4	1- 6	B.2	8	14	1	B.6	10
4	7-10	B.3	8	14	2	B.7	10
4	11-22	B.5	9	14	3	B.8	10
5	1	B.12	12	14	4	B.9	10
5	2- 5	B.10	11	14	5	B.10	11
5	6-11	B.13	12	14	6- 9	B.11	11
5	12-15	B.4	9	14	10-12	B.14	13
5	16-19	B.15	13	14	13-14	B.9	10
5	20-23	B.1	8				

Table 4.1: Action Rule references to DEMOSL 3.7

4.5.3.1 Ontological-Metamodel

So, for the AM, we need some property types between the action rules and the facts that come into existence or that are referenced from e.g./ action rule conditions. Therefore, we added property types for the 'then', 'else', and 'with-clause' specifications. Also, the property references have been transformed into property types which connect to other parts of the model. These property types should be combined with the attributes in action rule to model the complete AM.

4.5.3.2 Action Rule References

In appendix C.5 we have listed a few dozens of published references of action rules. These rules violate the proposed grammar in one way or another. The common reasons for violating the grammar are:

- Common mistakes that are introduced by having no automated checks on the grammar. These include a missing semi-colon, mistaken use of 'for' versus 'of' keyword, 'a' instead of 'some', etc.
- The removal of not needed assessment conditions. If the <no specific condition> condition appears, the proposed grammar leaves out the empty condition as well as the keyword.

- Addition of [] to mark the language part that needs to identify an attribute. This enhances the readability, for humans and for the parser.
- Value references that are written in language instead of in numbers, and often without units of measurement.
- Changes of [] to () in case of a transaction step reference.
- The usage of a condition expression in a with-clause statement. This is not allowed according to the grammar.
- Wrongly placed conditions in the truth clause that should be in the while clause.
- Otherwise formulated sentences that do not comply with the reasoning form.
- Date references (Today, Now, Month) that are used as if they were values. The grammar does not allow for that type of reasoning with named values.
- 'For each' references were not in the original grammar. All scripts containing the for each (in the while, with or action clause) have been rewritten.
- A lot of scripts were missing product references (e.g. is the transporter) that have to be added (e.g. of Transport).

It has been noticed that the usage of capitals were not consistent throughout the action rule specifications. We have decided to match the names on the referenced objects in the other aspect models instead of keeping strict references in the grammar.

4.5.3.3 Reserved Words

Grammar is build of words. These words have to appear in a particular order. Some words have a special meaning at a certain location in the grammar. When words have a very specific meaning and should not be confused with other words at specific locations, they can be excluded from other uses. Most programming languages have the reserved words 'if', 'then', and 'else'. The original and the proposed grammar for DEMO 3 have the reserved words as listed in appendix B.1 and C.3, respectively. It is worth noting that the original grammar of DEMOSL 3.7 has more reserved words than the proposed grammar.

Though no reserved words are really needed in the proposed grammar when using the right distinctive characters, certain words can cause some misunderstanding of the meaning of the sentence. Especially when the grammar lexar words are used in a place where they can have a double meaning (e.g. `the tire of car of rental` can be understood as `the [tire] of [car] of [rental]` which will result in an error or `the [tire of car] of [rental]`). To use lexar words in this context the parser can be helped by putting square brackets around the variable where needed. Some examples have exactly this adjustment (e.g. C.5.1 on page 24))

The exchange model has no reserved words because all the characters can be used within the internal coding of an Extensible Markup Language (XML).

4.5.3.4 Base Types

Base types are the core of the grammar and of the exchange model. Though in itself they are meaningless, we need them for other parts of the metamodel. The listing is used in all kinds of rules and is added for reference purposes in listing C.4 and listing C.5.

The exchange model uses the base type specifications of XSD and the names and id's that are defined in D.12 and D.8, respectively.

4.5.3.5 Element Id's

The DEMOSL 3.7 specification has six rules concerning the identity format of a specific element. Looking at the entry points of the formats, these are not used in the original grammar. Though we have created extended formatting of these identifiers, we will copy them as they do specify the original formatting of the element identification. The original formatting is specified in listing B.2, whereas the proposed grammar specification can be found in appendix C.1.
The Exchange Model definitions for the element Id's have been described in D.10 and match the grammar rules for the equivalent Ids.

4.5.3.6 Product Kind Formulation

The product kind formulation describes the result of the transaction kind. This is a way of describing the product in the past tense. Like element Id's, this rule is not directly used in the grammar. It could be used, in an adapted rule, to verify the formulation of the product kinds. The original specification is in B.4. The Exchange Model does not use the rule for the product kind formulation.

4.5.3.7 Element Names

The original and proposed element names of the DEMOSL elements have been described in the rules in listing B.3 and C.2, respectively. Both sets of rules are not used in the grammar directly. This causes some variations in naming conventions for attributes, objects, and other elements. The proposed grammar does not distinguish these elements as they are to be related to the original elements in the other aspect models. The Exchange Model does have the name conventions for the element names and can be found in appendix D.12. The naming convention for elements is checked in the Exchange Model.

4.5.3.8 Grammar Entry Point

The mainline in the grammar is the 'action rule specification'. Here the event, assess, and response parts are distinguished. The original grammar start (listing B.6) is equivalent to the proposed grammar (listing C.6). The proposed grammar has an extra name part, the name of the rule, equivalent to the information in the XSD (listing D.114). We added the rule name to have distinguishable names of rules in our model.

4.5.3.9 Event Part Specification

The original grammar places the with after the while clause. This means that the agendum gets two lists of with parameters which can be indistinguishable. We reversed the order

of these clauses and added the with-clause of the agendum-clause within this agendum-clause. This is listed in B.7 and C.7. The exchange model mimics the order of the grammar in listing D.115.

4.5.3.10 Agendum Clause Specification

The original grammar stops at the perfect tense intention. In the examples, it can be seen that a transaction step reference is also present. This is now addressed by the proposed grammar. The step reference and the with-clause are added to the grammar in the agendum-clause and both are listen in B.8 and C.10.

4.5.3.11 While Clause Specification

The original grammar for the while clause has a single with-clause. It also misses the transaction step reference that is used in the examples. The proposed grammar adds multiple with-clauses to a while clause and references the transaction step. This is listed in B.10 and C.13.

4.5.3.12 With Clause Specification

The with-clause is, in fact, a reference to the entity and attribute (with even the data type) that is being used for the transfer of information between initiator and executor (and vice versa). This information can be written in natural language (listing C.11) or in an exchange model with references to the specific entity and attribute (listing D.123). The references are created using the GUIDs of the specific entity or attribute to make sure that double naming will not reference the wrong attribute. Unfortunately, this cannot be specified in the grammar.
The original grammar for the with-clause is similar to the proposed grammar. The only difference lies in the object variable that is not defined in the original grammar for the property kind formulation.
Deduced from the examples of exiting ARS, we extended the property kind formulation to have the 'some DAY' and other property variable references in the grammar. The word 'some' refers to the reference-property type between two entities, with the attribute referencing the property type alias. The property kind formulation '**the** son **of** Person **is some** PERSON', or rather 'the [son] of [person] is some [person]', is equivalent to the self-referencing property type on Person being called 'son' as visualised in fig. 4.31.

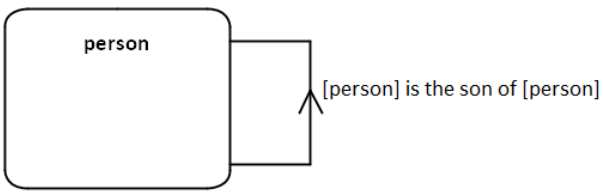


Figure 4.31: The meaning of some

4.5.3.13 Assess Part Specification

The original grammar for the assess part uses the <no specific> condition where the clause cannot be specified in terms of a list of fact kind formulations. The proposed grammar uses the same specifications for the assess-part as the original grammar. This is listed in B.11 and C.8.

4.5.3.14 Fact Kind Formulation

The original grammar for the fact kind formulation (listing B.12) makes a distinction between property, attribute and product kind formulations. The proposed grammar uses the same specifications for the fact kind formulation (listing C.15) except for the product kind formulation. This formulation is not used in practice to describe any references.

4.5.3.15 Attribute Kind Formulation

The original grammar for the attribute kind formulation (listing B.13) has three conditions that can be used. Some examples do not conform to these conditions and therefore cannot be expressed with the original grammar.
The proposed grammar uses more complete set of comparison possibilities. The 'and' and 'or' operators, that are used in the examples are added. Also the 'in' comparator is added (listing C.16). The original attribute variable was insufficiently specified and is replaced by the property variable that can specify an attribute of an entity in a set of values.

4.5.3.16 Response Part

In the studied examples, the cardinality of the property types between two TPSKs were used. This specification has been added to the proposed grammar. Also, the proposed grammar has the ability to specify multiple actions in the then and in the else part of the action rule response. Both original and proposed grammar are listed in B.14 and C.9, respectively.

4.5.3.17 Intentions

Intentions are the verbalisations of the GSK steps for the TPSK. The original grammar contains the past and perfect tense verbalisations (listing B.15). In the proposed grammar we added the abbreviations of those intentions to accommodate for the use of the TPSK in the when clause (listing C.17).

4.5.3.18 Entity Variables

The entity variable in the original syntax (listing B.5) has been superseded by the general naming of variables (listing C.18). The naming of the variables and the connection to other parts of the model can be deducted.

4.6 Implementation Metamodels

During practical application of DEMO models, we came to the conclusion that some additional diagrams were needed to fully represent the model.
The original DEMO metamodel did not allow for models to "grow" from the different aspect models, in the sense that the consistency rules would require the model to always be complete as a whole (so, including all aspect models). Therefore adjustments to the metamodel needed to be made to allow for such flexibility, while still enforcing (at the end of the modelling process) the overall consistency.

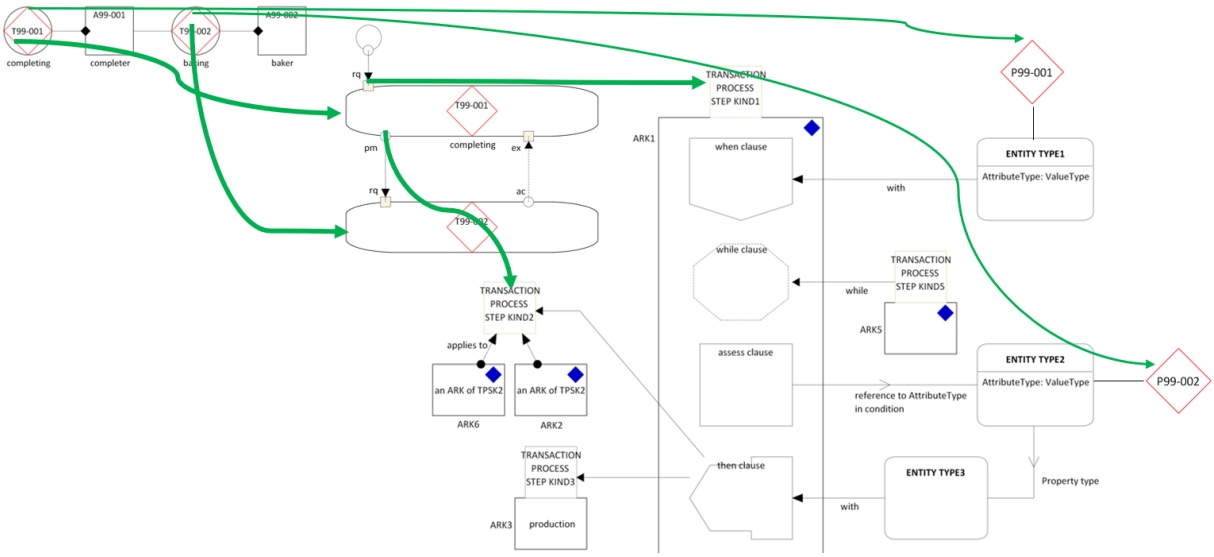


Figure 4.32: CM-AM flow

In practice, organisation models that resulted from the OER analysis often raised questions regarding the *origins* of the included transaction kinds. More specifically, a cross-reference from the elements in the organisation model to the original OER analysis was missing. To support this kind of cross-reference, we have added the interview and interview-line concept to the metamodel. By registering every aspect of the OER analysis as a connection from the interview-line to the elements modelled from that line information we have created a cross-reference from the source to the final model.
Another finding is that in practice, there was a need for DEMO models to be related to their existing / planned implementation. For instance, a "serving" connector was introduced that could be used to connect e.g. DEMO model elements to e.g. application components in the ArchiMate's Application layer. With this connector, one can, for instance, point to application components that implement the transaction kind or TPSK. Another example of the need to be able to include more of the implementation context involves the introduction of an actor (type) that aggregate actor roles. Such actor types correspond to job functions and, therefore, combine all competences of the actor roles that are aggregated. This can be used for HR implementation information.

4.7 Property Types

Tables below describe all allowed Property Types between the objects of the metamodel (4.2), as well as all Property Types and elements that are allowed to be used in diagrams (4.3).

To → From ↓	ETK	ATK	EAR	CAR	ET	CET	TPSK	AR
ETK	-	c	e	ce	-	-	-	-
ATK	-	-	-	-	-	-	-	-
EAR	ia	a	-	c	-	-	-	-
CAR	ia	a	-	c	-	-	-	-
ET	o	-	-	-	xsrg	-	-	-
CET	o	-	-	-	-	-	-	-
TPSK	c	-	-	-	-	-	iy	tlwh
AR	-	-	-	-	W	-	-	-

Table 4.2: Element Property Types

Diagram → Entity Types ↓	OCD	PSD	TPD	OFD	ARD	RHD	AFD	..
ETK	X	X	X	X	-	-	-	-
ATK	X	-	-	-	-	-	-	-
EAR	X	X	-	-	-	X	-	-
CAR	X	X	-	-	-	X	-	-
ET	-	-	-	X	-	-	-	-
CET	-	-	-	X	-	-	-	-
TPSK	-	X	X	-	X	-	-	-
AR	-	-	-	-	X	-	-	-
Property Types ↓								
[c] contained in	-	-	-	-	-	X	-	-
[o] concerns	-	-	-	X	-	-	-	-
[i] initiator	X	X	-	-	-	-	-	-
[e] executor	X	-	-	-	-	-	-	-
[a] access to bank	X	-	-	-	-	-	-	-
[s] specialisation	-	-	-	X	-	-	-	-
[r] aggregation	-	-	-	X	-	-	-	-
[g] generalisation	-	-	-	X	-	-	-	-
[t] then	-	-	-	-	X	-	-	-
[l] else	-	-	-	-	X	-	-	-
[w] while	-	-	-	-	X	-	-	-
[h] when	-	-	-	-	X	-	-	-
[W] with	-	-	-	-	X	-	-	-
[x] excludes	-	-	-	X	-	-	-	-
[y] wait	-	X	-	-	-	-	-	-
[f] role of	-	-	-	-	-	-	X	-

Table 4.3: Diagram Entity and Property Types

4.8 Model Extensions

Because the DEMO notation is never finished and the links to other notations will require multiple adjustments, the metamodel needs to remain extendable. This topic is further elaborated upon in the ExtendedInfo part and can be found in listing D.124.

Chapter 5

Tool Implementation

For this research, we selected a tool to implement the DEMO metamodel. In this chapter, we outline the tool selection process and the initial creation of the various metamodels in the tool. In the closing section of this chapter, we will further describe the verification mechanism that was built on top of the metamodel.

5.1 Tool Selection

In the search for an tool to create DEMO models we have looked for building from scratch, using an existing tool, or extending a tool. At that time, building from scratch was not feasible and we have only looked at the other two options. Therefore, selecting a tool from a set of available tools needs guidance. A tool can have a user preference based on look and feel, on creator affinity, or just on the name. We needed to define some objective criteria that guided us in choosing the right tool. In this section, we will describe the literature we used and our outlined arguments for choosing this particular set of requirements.

5.1.1 Requirements for Tool Selection

For our research, we selected an enterprise-grade tool (framework) to implement the DEMO metamodel based on the criteria as listed below. We do not pretend that this set of criteria is complete or adequate for all users of the final modelling tool. However, for the purposes of our research, we deemed these to be appropriate. We used this set of criteria to reduce the selection of potential tools and to finally come up with a usable tool, fulfilling the most relevant objective criteria. Additionally, an earlier version of these criteria, as reported in [6], has also already been used by other researchers [64].
In the selection of the tool (framework), the following overall criteria were used:

1. The tool must be able to support the executability of the final model;
2. The tool must be able to support all the ways of modelling and thinking as described in the DEMO methodology;
3. The tool must be able to support the option to interchange with VISI (this originally stands for 'Voorwaarden scheppen voor de Invoering van Standaardisatie ICT in de bouw') models when applicable;

4. The tool must be able to check the logical, internal correctness of the model;
5. The tool must be able to have an efficient storage of the model;
6. The tool must be able to support the validation of a model with relevant stakeholders.

From this list, we then defined the requirements for the tool.

5.1.1.1 Defining Requirements

Modelling an organisation with DEMO requires the use of all four DEMO aspect models. This results in the first requirement that is defined as:

Requirement 1 *The tool must be able to support the creation of all four aspect models of DEMO.*

The specification language of DEMO [1] states the high-level ontological-metamodel that needs to be supported, as well as the visual representations of the different models and diagrams. Although the DEMOSL is not complete (yet), as is shown and elaborated on in Chapter 3, it certainly outlines the minimal descriptions and guidelines to visualise DEMO diagrams and defines the minimum set of information to create a ontological-metamodel in order to describe the internal relationships of a DEMO model. To ensure compliance with these specifications, our second requirement states:

Requirement 2 *The tool must be fully compliant with the DEMO metamodel, as specified in DEMOSL.*

One of the research topics in the field of DEMO is the potential to integration and combination of other modelling methods and notations [51, 63, 79, 90]. When we follow the recommendations outlined in these publications, the best way for a tool to support the combination of modelling methods is to have a tool that also supports various modelling methods and notations. This has such a practical benefit, apart from supporting further research, that we added it as a third requirement:

Requirement 3 *The tool must be able to support multiple modelling methods and notations.*

When going through DEMO models in published scientific papers, we found incompleteness, structural incorrectness, and syntactic failures in the various models [79, 65, 91]. To solve this problem, the DEMO model should be verified against the specification language. To conduct such a verification, requires a considerable effort as it involves a large body of correctness rules. For example, the Xemod tool (see Section 5.1.2) already has 23 business rules published and implemented for the CM[1]. In Section 5.3, we will elaborate further on all verification rules that are needed to verify the whole model. When the task of verification takes more time than the modelling itself, we can also accidentally introduce defects in the model. To compensate for the effort, the tool should be able to automate as much verification during modelling as possible. This leads us to the fourth requirement:

[1] [92] - J. Vos. *Business modeling software focused on DEMO; http://wiki.xemod.eu.* 2011. URL: `http://wiki.xemod.eu`

Requirement 4 *The tool must allow for automated model verification against the DEMO metamodel, DEMOSL.*

The DEMO methodology itself does not prescribe the modelling order of the DEMO model such that it is the only correct order of modelling [25]. The OER method only determines the components that need to be modelled and how these modelled components need to be connected. As a consequence, the model could not be built solely on using req. 4. Therefore, we need an extra requirement for the tool that allows for variation in modelling order:

Requirement 5 *The tool must allow for model verification for any modelling order in which a model is created.*

Although four metamodels were specified in the original DEMOSL, in Chapter 3 we created the integration of these four metamodels into a single, integral, complete metamodel. This single metamodel allows for easy subsequent model verification. Using this metamodel as base for a single repository allows the reuse of components of the model in other aspect models. We found the usage of a single metamodel and repository to be useful, and therefore, we state the sixth requirement:

Requirement 6 *The tool must be able to keep produced models in an integral metamodel and repository.*

Modelling tools have been around for several decades, and still, there is no common modelling tool available that could apply for the de facto standard for DEMO. The tools that are used for modelling DEMO seem to be missing some elementary aspects or are not capable of modelling all the different aspect models. In Section 5.1, we have looked at a selection of currently used tools. Although there might be more tools that are in use or that are capable to model organisations, we limited our research to the described set of potential tools. We conjecture that existing tools are not optimal for DEMO in the standard available version. Though we have covered the aspect models of DEMO in req. 1, more facets of this methodology are important. Therefore, we introduce our seventh, requirement:

Requirement 7 *The tool must allow for extending the modelling tool capabilities when this is deemed necessary for full support of DEMO.*

A tool that can capture all DEMO related models and tables, and possibly even other related modelling techniques, may sound ideal. However, such a tool *also* needs to be viable in an enterprise context, which means that the enterprise-grade criteria as discussed in section 2.8 become crucial.
Therefore, finally, considering the need to support the use of DEMO in larger enterprises, a selected tool needs to be *enterprise-grade*:

Requirement 8 *The tool (framework), and its provider, must be enterprise-grade (see section 2.8).*

Finally, though usability of tools is an important subject, we have not included it in our considerations. On the one hand, usability from a user-interface point of view is highly subjective. Where one person might like mobile devices based on Apple interfaces, another person might like Android interfaces, while both do the job just a bit different. On the other hand, in the context of modelling tools, usability is also strongly influenced by the notation [41] as used in a modelling language / method. Since we have taken DEMO as-is as our starting point, it would not be relevant (within our research) to consider the usability of the DEMO.

5.1.1.2 Additional Requirements

In the previous section, we discussed and stated the requirements that will define the suitability of the tool for DEMO modelling. Next to these requirements, some non-scientific requests come into mind when selecting a suitable tool. For the sake of overall completeness, we will mention them and list them as additional requirements. We will, however, not use them in our requirement validation process.
For any methodology to become widely adopted by professionals in the field, supporting modelling tools are of paramount importance. Therefore, selecting a tool that already has a profound market penetration is better for the purpose of wide adoption. The soft requirement, therefore, is:

Requirement 9 *The tool must have a substantial market penetration.*

Substantial market penetration of a tool is important to create a wide adoption, but new consultants and scientists who use a tool are of course educated first. When a tool is available in the early stages of the careers of scientists or consultants the use of the tool in their professional life becomes more likely when the tool is also proven to be helpful for them. This is expressed as:

Requirement 10 *The tool must be available for educational purposes at a low cost.*

Next to these requirements, the DEMO metamodels need to be "built" on top of the MU theory (section 2.3), which is a foundational part of the DEMO methodology [9] and was known in a previous iteration of DEMO as *Factual Knowledge* [25].

5.1.2 Current Tools on the Market

In the search for modelling tools that support modelling in DEMO and that comply with all the requirements as listed above, we have been able to find ten potential candidates. This initial selection was based on (1) existing experiences with tools within the community of DEMO practitioners, (2) potential or shown openness of the tool (framework) and / or tool vendor to accommodate DEMO models. The resulting ten candidates were thoroughly studied and checked against the specified requirements as discussed in section 5.1.1.
We do not pretend to have conducted an exhaustive search for, and evaluation of, all available tools on the market. As discussed in the introduction, the lack of tooling was

found to be a hampering factor in the further uptake of the DEMO method. A phenomenon, which we also observed to be the case in the first years in the existence of the ArchiMate [42] standard. Basically, a chicken-and-egg problem. For strategic reasons, we therefore primarily focused on finding a suitable tool among the enterprise (architecture) modelling tools that were already known-to, and/or used-by, members of the community of existing, and *potential*, DEMO users. The assumption underlying this strategy is that selecting an already used, or at least known, tool would also result in a quicker acceptance of a DEMO tool within the (initial) target audience. The latter will then, hopefully, also trigger additional (large) tool vendors to follow the example and include support for DEMO in their tool/framework as well. This is a pattern which we think to also have observed in the uptake of the ArchiMate standard and associated tooling.
As such, building a tool from scratch was also not considered to be a desirable option, also when taking the enterprise-grade requirements into consideration. In alphabetical order, we present the results of our analysis below.

Figure 5.1: Connexio

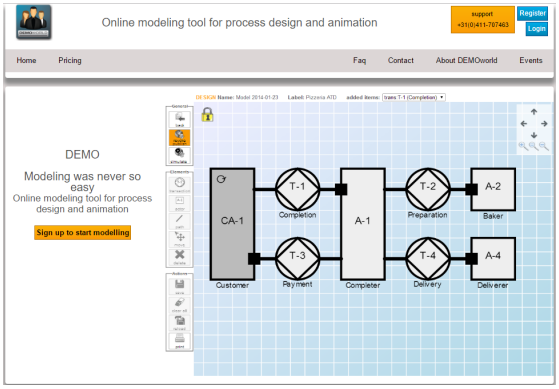


Figure 5.2: DemoWorld

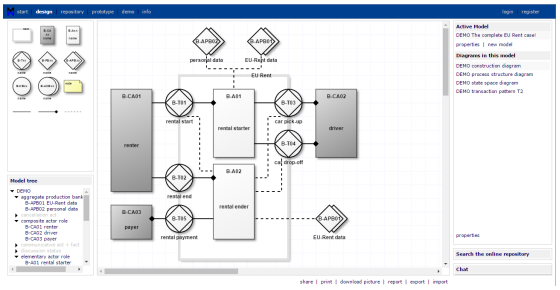


Figure 5.3: ModelWorld

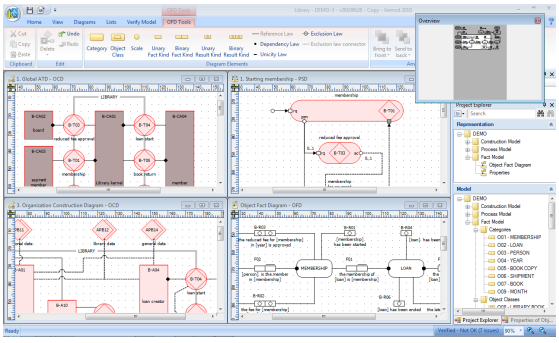


Figure 5.4: Xemod

ARIS by Software AG[2] is a business modelling tool. With the use of the ArchiMate modelling possibilities (req. 3), the set of objects within this tool has been extended with a transaction and actor role to allow for OCD modelling (req. 1). This tool is available and commonly used equivalent to (req. 9). This tool is free of charge for 6 months to make it suitable for educational purposes (req. 10).

[2] [93] - Software AG. *ARIS http://www2.softwareag.com/*

CaseWise Modeler by CaseWise is a business modelling tool that collects and documents the way in which organisations work. This tool has an Application Programming Interface (API) and can, therefore, be extended further. Information about the extent of the API is not available[3]. This tool can provide multiple diagrams, e.g. the Entity Relation Diagram (ERD), Calendar, and Process simulation equivalent to (req. 3). The diagram features of this tool can be used to create DEMO diagrams except for the AM, which is too complex for this tool (req. 1). The repository can contain standard and extended objects for one model (req. 6). Regretfully, no verification business rules can be added to the tool. Finally, the tool is available and commonly used (req. 9).

Connexio Knowledge System by Business Fundamentals[4] documents the representation of the current organisation implementation using the OCD diagram (req. 1). It also stores ARS and WISs to enable people to manage and understand their job knowledge. It does not support all aspect models of DEMO. It is proprietary software and for internal use only.

DemoWorld by ForMetis[5] is an online modelling tool for DEMO processes (req. 1). It does verify some business rules of the OCD (req. 5) and also allows for OCD modelling and process animation. The tool cannot model all aspect models and has no built-in verification options. It is available for commercial use. Students pay €50 per 6 months, professors are able to get free usage for 6 months.

EC-Mod by Delta Change Consultants can visualise organisational flaws in the transaction kinds and actor roles, using a modified OCD (req. 1). This modified OCD shows organisation flaws and can also be verified (req. 4). It was presented in 2016 but is for internal use only.

Sparx Enterprise Architect (SEA) by Sparx is an analysis and design tool[6] for Unified Modelling Language (UML), SysML, BPMN, and several other modelling techniques (equivalent to req. 3). It covers the process from the analysis of requirements gathering through to model design, building, testing, and maintenance. Furthermore, it allows for the creation of implementation metamodels to model one's objects, connections and business rules (reqs. 5 and 7). The repository stores in this tool all objects to enable reuse on multiple diagrams (req. 6). It is available and commonly used (req. 9). The tool has a one-time fee of €60 for individual academic licenses (req. 10).

---

[3] [94] - CaseWise. *The Casewise Suite and Casewise Modeler; http://www.casewise.com/product/modeler*. 2016. URL: `http://www.casewise.com/product/modeler/`

[4] [95] - T. Severien. *Business Fundamentals - Verbeteren vanuit de essentie; http://www.businessfundamentals.nl/*. 2016. URL: `http://www.businessfundamentals.nl/`

[5] [96] - Formetis. *Online modeling tool for process design and animation; https://www.demoworld.nl/*. 2017. URL: `https://www.demoworld.nl/Portal/Home`

[6] [97] - Sparx. *Enterprise Architect https://www.sparxsystems.eu/start/home/*. 2017. URL: `https://www.sparxsystems.eu/start/home/`

ModelWorld by EssMod[7] can support ArchiMate, DEMO, BPMN, UML, and Mockups (req. 3). The repository contains all models (req. 6). It can visualise three aspect diagrams of DEMO (req. 1), but these can not be verified using business rules. Though the model can be simulated, no further extensions can be written on this tool. The tool has been free of charge (req. 10). Unfortunately, this tool is currently no longer available.

Open Modeling by Santbrink is a multimodel tool that supports Flowcharts, Integration Definition (IDEF) scheme, Application landscape, DEMO, ArchiMate, Use Case diagram, Component diagram, State diagram, ERD, Data Flow Diagram (DFD), and Screen sequence diagram.

uRequire Studio by uSoft is a requirement management tool[8] that has the option to add OCD (req. 1). The requirements repository cannot store DEMO models. As a consequence, diagrams are stored and only connected to the requirement definitions by object description. No other aspect diagrams of DEMO are available. The tool also supports BPMN diagrams (req. 3) and is available for commercial use.

Visio by Microsoft is a drawing aid with templates for DEMO resembling the graphical representation (req. 1). Multiple diagram types can be made in a single file (req. 3). Visio does not have a repository of objects. Diagram verification is possible using programming in Visual Basic for Applications (VBA) (req. 7), but diagram objects cannot easily be related. The tool is available and commonly used (req. 9) for modelling. A Visio licence for students is available for €50 (req. 10).

Xemod by Mprise (fig. 5.4) is a tool that has the ability to integral model three DEMO aspect diagrams within a single Scope of Interest per project file[9] (req. 1), which might result in an inconsistency between multiple scopes of interests, scattered in models. Business rules can be used to verify the model on consistency between several elements (reqs. 4 and 5). Furthermore, the tool can show the OCD, PM, and FM as diagrams and lists (req. 6). Unfortunately, this tool is no longer available on the market.

[7] [98] - Lambertus Johannes Hommes. *ModelWorld; http://ModelWorld.nl.* 2015

[8] [99] - uSoft. *URequire Studio; http://www.usoft.com/software/urequire-studio.* 2016. URL: `http://www.usoft.com/software/urequire-studio`

[9] [92] - J. Vos. *Business modeling software focused on DEMO; http://wiki.xemod.eu.* 2011. URL: `http://wiki.xemod.eu`

	DEMO(1)	DEMOSL(2)	Multi-model()3)	Verifiable(4)	Business Rules(5)	Repository(6)	Extendable(7)	Enterprise-grade(8)	Total score	Cloud/Local	Simulation	Available
ARIS	C---	–	✓	–	–	–	–	✓	3	L	–	✓
CaseWise Modeler	CP-F	–	✓	–	–	✓	–	✓	4	L	–	✓
Connexio Knowledge System	C---	–	–	–	–	–	–	–	1	L	–	-
DemoWorld	C---	–	–	✓	✓	–	–	–	3	C	✓	✓
EC-Mod	CP--	–	–	✓	–	–	–		2	L	–	-
SEA	----	–	✓	–	✓	✓	✓	✓	5	L	–	✓
ModelWorld	CP-F	–	–	–	✓	✓	–	–	3	C	✓	-
uRequire Studio	C---	–	–	–	–	–	–	–	1	C	–	✓
Visio	C--F	–	✓	–	–	–	✓	–	3	L	–	✓
Xemod	CP-F	–	–	✓	✓	✓	–	–	4	L	–	–

Legend for DEMO coverage: C = Construction Model, P = Process Model, A = Action Model, and F = Fact Model.
Total score = sum of ticks, plus 1 if at least the Construction Models are supported

Table 5.1: Summary of the findings on available tooling software

Table 5.1 provides a summary of our findings on available tooling software. As can be seen from the table, we conclude that DEMOSL has not been completely implemented in any tool available yet. More specifically, we conclude that no current tool can integrally capture all models and diagrams needed for the methodology as a whole. In principle, this would have brought us to an impasse in the sense of tool builders waiting for a further increase of the use of DEMO, while potential DEMO users wait for enterprise-grade tool support. To move beyond this impasse, we decided to extend a commonly used framework. In line with this, the scores suggested that SEA would be the best candidate to fulfil the requirements, by extending the tool with DEMO. SEA is broadly used, supports multiple models, can apply business rules, has a repository, is extendable, and is already used for architecture modelling. More specifically, the SEA platform allows for the implementation of new/additional modelling languages by means of C# based extensions in combination with "shape scripts" for the graphical shapes. Therefore, this tool framework was chosen to be configured with the DEMOSL metamodel and implement the DEMO methodology as precisely as possible.

5.2 Creating the Metamodel in SEA

The modelling tool SEA allows for extending the basic UML model with the extender's concepts. These extended concepts are called profiles. In the next sections we will explain how we have built the DEMO metamodel in and for SEA. During our research, we have used the SEA tool version 13.0.1310 up to 14.1.1429.

5.2.1 UML Typeset

The modelling tool SEA has a built-in metamodel base consisting of UML data types. By creating a new so-called profile, these data types can be extended with one's own "extended concepts". Many metamodels in the SEA modelling tool are using these UML data types as a base for their own model (e.g. ArchiMate, BPMN). Therefore, the used UML types for our DEMO metamodel in SEA are *Class* for entity types and *Association* for the relations between entity types.

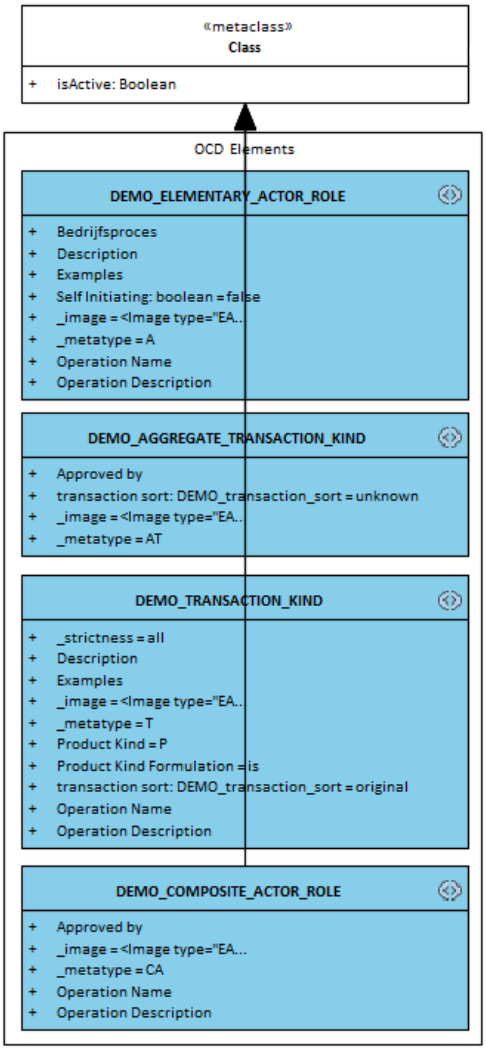


Figure 5.5: OCD profile, standard visualised by SEA as class extensions

5.2.2 Extending Types

To extend a model in SEA, one needs to define the model in three distinct profile types: the stereotype ≪*profile*≫, the ≪*toolbox profile*≫ and the ≪*diagram profile*≫. In this section, we will only discuss the extending of the SEA metamodel for the aspect model CM and aspect diagram OCD.

5.2.3 Stereotype Profile

To be able to model the stereotype ≪*profile*≫ of the CM, we needed to create an implementation of the DEMOSL concepts into the existing SEA modelling options. SEA uses a ≪*stereotype*≫ for each potentially visible object. Therefore, the ontological metamodel of the CM needs to be reduced to visible entity types. In the metamodel of the Construction Model (CM) of DEMOSL 3.7, as shown in fig. 3.1, we can see that the ontological

metamodel contains six entity types. The entity type IFK or EVENT TYPE is the link between the CM and the FM. It is not displayed on the OCD and can be left out of the SEA metamodel for the CM, just as the FACT KIND. Earlier, we already mentioned the missing Scope of Interest Boundary (SIB) in the CM metamodel. We decided to use the CAR concept instead and remove the IFK which reduced the number of ≪*stereotype*≫s concerning the CM to four entity types, as seen in fig. 5.5.
We extended the ≪*meta-class*≫ Class for the EAR, CAR, ATK, and TK elements and added properties, attributes, and shape scripts (as explained in section 5.2.6) to the four stereotypes. Each stereotype has a 'meta-type' property showing the default name of the instantiated model element. The how and why of other properties of each stereotype are available on request and will not be further discussed in this section. We also extended the ≪*meta-class*≫ Association to create connectors between the elements.
We defined the name of the instantiations of the discussed ≪*stereotype*≫ in order to distinguish the metamodel, model, and visualisation.

Definition 1 *An Element is an instantiated Class extension* ≪stereotype≫ *in the repository.*

Definition 2 *A Diagram Element is a Link to an Element, and visualisation placed on a diagram.*

Definition 3 *A Connector is an instantiated Association extension* ≪stereotype≫ *in the repository.*

Definition 4 *A Diagram Connector is a Link to a Connector and visualises the connection between two Diagram Elements on a Diagram.*

5.2.4 Diagram Profile

Aside from the ≪*stereotype*≫s, as defined in section 5.2.3, we needed to define the diagrams where element links reside on (see fig. 5.8). For the diagram OCD we have extended the ≪*meta-class*≫ Diagram_Logical into the ≪*stereotype*≫ 'DEMO_OCD' and used the 'diagramID' to identify the specific diagram when instantiated. The associated toolbox to create elements will be discussed in section 5.2.5.

Definition 5 *A Diagram is an instantiated Diagram_Logical* ≪stereotype≫ *in the repository.*

The diagrams ≪*stereotype*≫ will function as a unique identification of the diagram type OCD. Not only does this allow for verification of the element links on the diagram in a later stage, but it can also be used for selecting the right visualisation of the element link on the diagram, as diagrams with element links are only a viewpoint to the model as it should be. We needed to define all diagrams of DEMO in the ≪*diagram profile*≫.

5.2.5 Toolbox Profile

In SEA, every diagram can be equipped with a toolbox with elements (see fig. 5.6) that displays the new elements and connectors which are meant to be used for that diagram. The elements can be dragged and dropped into the diagram and become new elements in the repository as well as (visual) element links on the diagram. The visualised connectors (called "links" in the associated toolbox) can be selected and the connector link can be dragged and dropped between two element links. This creates the connector link and consequently the connector between elements. To model these options for the diagram, the ≪*stereotype*≫ of the diagram needed to be extended from the ToolboxPage element (see fig. 5.7). The toolbox standard only offers the available elements and connectors that are valid for the selected diagram in a list type visualisation. It does not restrict the elements that can be used on the specific diagram. These restrictions are implemented in another part of the SEA model. Available connectors can be suggested to the user for specific diagrams using Quick Links. At this moment in our research, we did not yet implement the Quick Links into the tool.

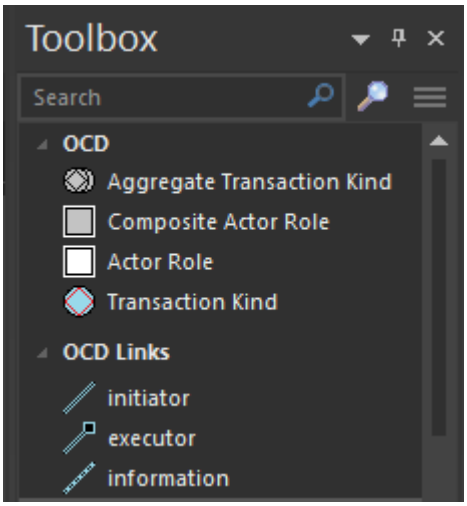


Figure 5.6: SEA OCD toolbox

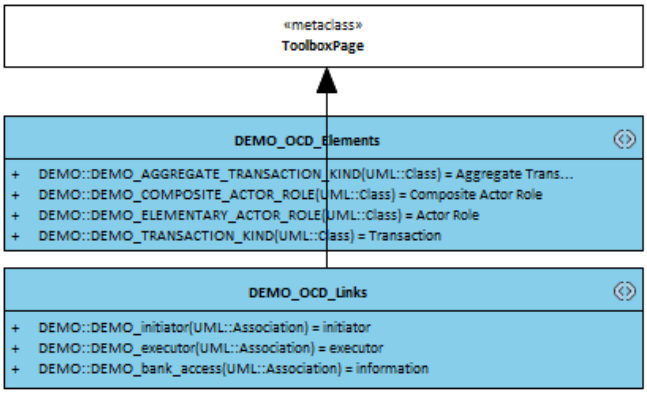


Figure 5.7: toolbox profile OCD

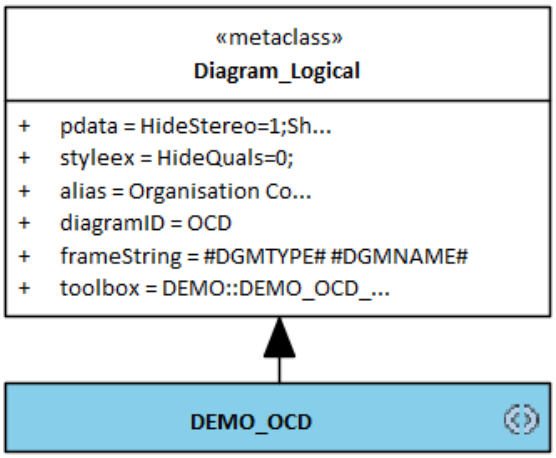


Figure 5.8: profile diagram OCD

```
shape OFDTransactionKind
{
    h_align="center";
    v_align="center";
    editablefield="name";
    fixedAspectRatio = "true";
    SetFillColor(255,255,255);
    roundrect(0,0,100,100,30,30);
    SetPenColor(0,0,0);

    polygon(50,50,4,40,0);
    print("#name#");
}
```

Figure 5.9: shape script OCD TK

5.2.6 Shape Scripts

SEA allows for each element link to be represented by a graphical shape. The creation of that shape is supported by the programming language *shape script*. *Shape script* has a syntax similar to C but has a limited set of available commands and limited functionality. The example script to draw a transaction kind shape is shown in fig. 5.9.
Within the shape scripts, it is possible to use some of the element properties or the containing diagram properties. These properties can be used in simple logic in order to determine the required shape to be drawn on the diagram. Limitations arise when shapes have different representations on different diagrams of the same type. When choosing the visualisation of a CAR, the programming features are too limited to either draw it as an internal rectangle and use it as Scope of Interest (SoI) or draw it as a grey-filled external rectangle automatically. The programming features do not allow for a context check of the visualisation. Therefore, these specific visual properties of the elements within SEA must be set manually by the user at the moment resulting in all instances on all equal diagram types having the same appearance.

5.3 Verification Model

The verification of all data rules, mathematical rules, and other model restrictions need to be automated in order to be useful to the DEMO modeller. In this section we describe a few of these rules. A more complete set of verification rules can be found in appendix F.

5.3.1 Model Verification

We implemented the verification of the DEMO model for multiple reasons. First of all, the elements and connections can be added to the SEA repository by third parties using the API. The use of this API entry path will bypass the necessary checking of the business rules because only User Interface (UI) events are available to be checked by the add-on in

SEA. The presence of a verification engine allows DEMO modellers to create valid models. Secondly, the DEMO model may be entered to the best ability of the modeller with all information available but is still incomplete. Without proper verification, the verification engine emphasises the gaps identified and allows for corrective measures. Lastly, the SEA tool has its limitations on keeping the diagrams within the specifications of DEMOSL upfront. Verification in the tool is checked afterwards. Therefore, the diagrams can also be corrected after new elements have been entered into the DEMO model.

5.3.2 Business Rules

The SEA tool has a flexible model extension feature but does not allow for a complete set of configurable business rules. Although some restrictions on inter-dependency can be made using e.g. the Quick Link feature, most restrictions and checks have to be made using the API. This API allows for checking business rules at UI events such as menu click, element creation (see fig. 5.10), deletion, and drag and drop onto a diagram canvas. We have used this API to implement business rules on relevant events.

```
public bool EA_OnPostNewElement(Repository repository,
          EventProperties info)
   {
     //Get element from the info
     int elementId = Convert.ToInt32(info.Get(0).Value);
     Element element =
       _repository.GetElementByID(elementId);
     ...
     Logger.Log("Created element {0}", ... );
     return false;
   }
```

Figure 5.10: New Element Handling C#

5.3.3 Test Model

The test DEMO model that we used to verify the DEMOSL within the SEA tool is the model that is described below in figs. 5.11 to 5.13.
The OCD contains these actor roles and transaction kinds:

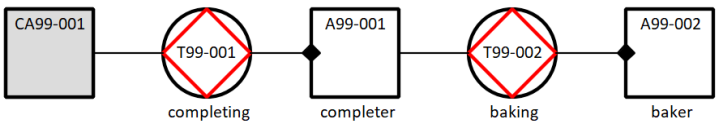


Figure 5.11: OCD Test model

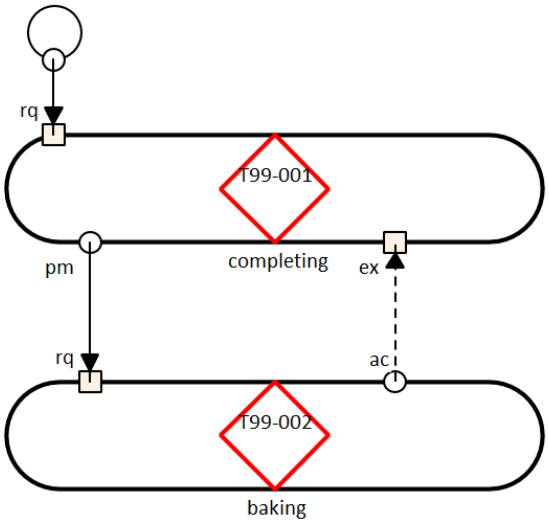


Figure 5.12: PSD Test model

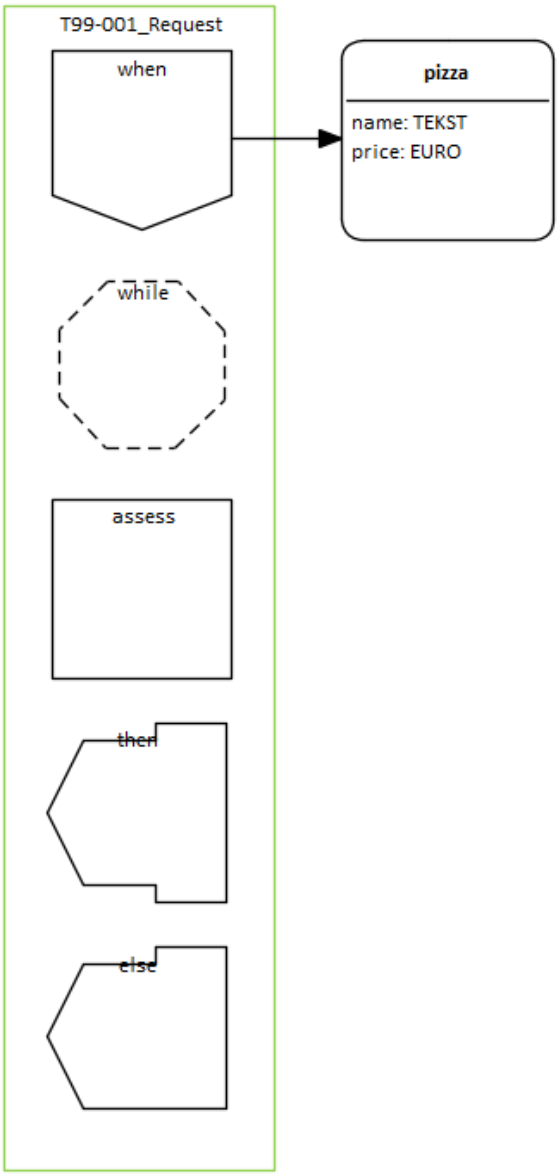


Figure 5.13: ARD Test model

5.3.4 Construction Model

The business rules that we have implemented for the OCD are:

1. Elementary actor roles may only be the executor of a single elementary transaction kind.
2. Every pair of actor role and transaction kind can only have a single connection of the initiator, executor, or bank access type.
3. On an OCD diagram, only ATK, TK, EAR, and CAR elements can be added.

All equations have been programmed into the verification metamodel (see fig. 5.15 part 6). These examples show how the program was built in order to comply with the DEMO

verification metamodel.

$$\mathsf{ActorRole}(x) \implies \mathsf{ElementaryActorRole}(x) \veebar \mathsf{CompositeActorRole}(x) \qquad \text{(E.5 ref)}_{\text{AR}}$$

```
actorrole = new List<BaseKind>();
actorrole.AddRange(m.ElementaryActorRoles);
actorrole.AddRange(m.CompositeActorRoles);
```

Listing 5.1: Actor role, joining composite and elementary in a list for verification (see eq. E.5)

All equation references we have programmed for the CM can be found in appendix F.1. The propositions of a specific type are implicitly programmed in the verification metamodel to make sure all List<>s are of the right type. The equation explicitly written in eq. E.5 is implicitly covered by building the lists as shown in listing F.6. The construction model is built of actor roles and transaction kinds. Equation E.2 is programmed as shown in listing F.7. The mutual exclusion equation of a TK (eq. E.6) and AR (eq. E.5) can be checked by the pieces of the program as outlined in listing F.8 and listing F.9, respectively.
We have split the verification of eq. E.7 and eq. E.8 in two separate verification rules. This is different from the logical representation but is compliant to the relational formulations of the XSD as outlined in listing D.79 and listing D.80, respectively.

5.3.5 Action Model

We needed to verify the action model with the logical metamodel and the exchange metamodel. Therefore, we have built five program parts (see fig. 5.14 for visualisation):

1. A verification metamodel programmed in code in the memory model (closely resembling the exchange model).
   This verification metamodel is restricted to the action rule name, the TK and TPSK on which the action rule applies, the entities involved in the with clause and, finally, the relationships between the entities involved. For testing purposes, we have used a simplified example.
2. A verbalisation generator has been programmed to create the ARS from the memory model.
   This generator takes the memory structure tree and follows the structure depth-first towards the end of the action rule, thereby writing the language. The tree navigation is in compliance with the action rule grammar. More structures are allowed in the memory model than in the grammar due to the inherent strictness of the memory model.
3. A model converter has been programmed to split the connected CM, PM, and FM concepts needed for the AM.

Although it would be more efficient to fetch the information directly from the original DEMO model, the assumed requirement of independence of the grammar module makes it necessary to create a new lookup structure for the model element.

4. We programmed an interpreter on top of the lexer-parser-listener of the created grammar that ultimately produces the exchange code.
   After lexing and parsing the input string, we are able to navigate through a tree which, again, follows the grammar structure. Now, instead of the language, we can fill the XML nodes into the XSD structure. Afterwards the XML nodes can be serialised in order to to create an XML string.
5. A direct conversion from the memory model to the exchange model.
   Serialisation of the memory model can also be done via the XSD structure directly. This step creates an XML from the memory model.

At the end of the path, we have two XML files that need to be identical for the AM part. The AM must represent the same information in all formats.

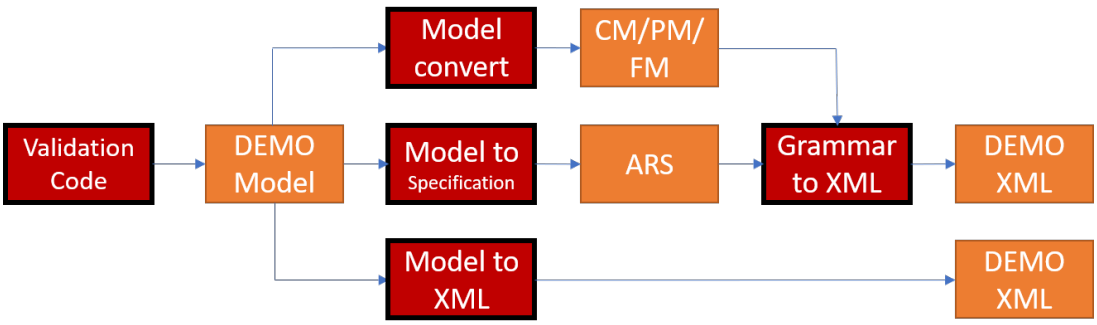


Figure 5.14: ARS Test path

Both XML files need to have the same content, and, therefore, no information loss can be detected during the conversion of the memory structure into the ARS language. As such, the information in the ARS has sufficient power to specify an action rule that can also be modelled in the memory model.

5.4 Result

Figure 5.15 shows the tool "in action", as well as some details of the tool's *implementation* on the SEA platform. In the latter, we see (numbers referring to the respective windows as displayed in the figure) how SEA provides a set of base classes (1) for elements, connectors, diagrams, and toolboxes. Each DEMO model element involves an extension of one of the base classes of SEA (2), while also having custom properties based on the DEMO metamodel (3). To enable various visualisations for different aspects, SEA provides simple scripts that enable the drawing of shapes (4). SEA provides interface "hooks", that are triggered when editing models in the graphical user interface, that can be used by add-on applications to provide functionality that is not available in SEA (5). The verification of DEMO models has been implemented in C# using these "hooks" (6).

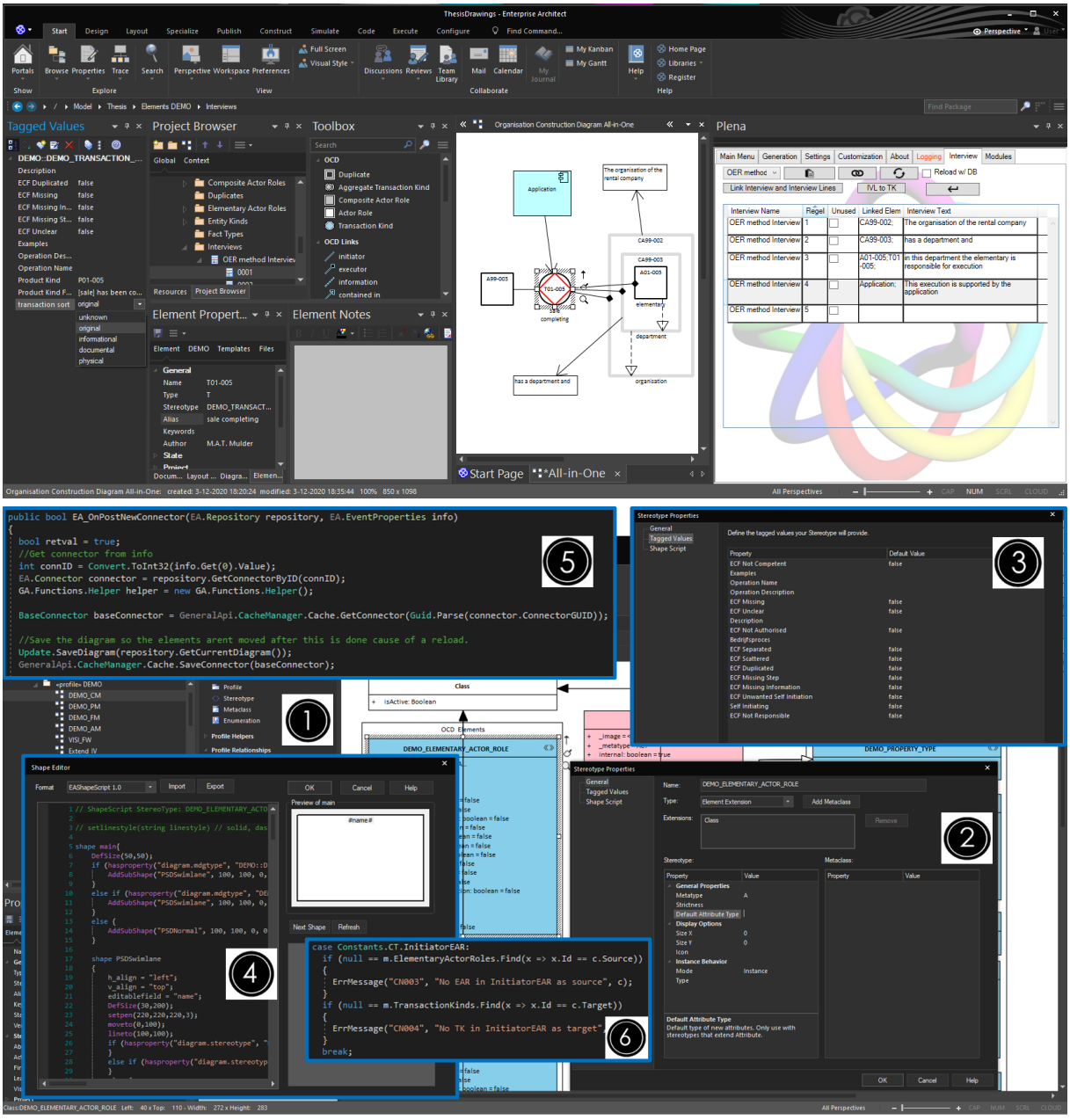


Figure 5.15: Tool in action, and its implementation

Part II

Implementing the Metamodel

Chapter 6

Practice: Model Iterations

Part of this chapter has been previously published in the paper for the EEWC 2019 conference[2].

6.1 Model Iterations

Over the past two years, we have created business models for five different organisations. The creation and presentation of these business models are described as separate iterations which vary in complexity, modelling effort, and modelling purposes. Even though the contexts among these models vary, the descriptions all start with the context of the particular iteration, the way of working during creation of the models, their presentation, the iteration adaptation during the modelling and discussions of these models. DEMO is a non-domain specific methodology that allows us to use the iterations throughout the various domains while the stakeholder types remain stable.
The iterations, which are labelled A-E, have been created in five different companies in the Netherlands. Iteration A, D, and E were used at logistic wholesale companies. Iteration B was created at a small property management company, whereas iteration C was created at a small call centre. We presented the business models to several types of stakeholders, which we distinguished in the following way:

- A developer builds a piece of software and uses models that follow the functional intention of that specific model(s).
- An operational employee is a business user that uses the models to reflect on his or her work.
- A director is a manager who is responsible for the efficient and effective operation of the organisation itself. One uses the models to get an (executive) overview.
- A C-level director (e.g. CEO, CFO, CIO...) is a manager who is responsible for parts of the organisation that a director manages. One uses the models to optimise one particular aspect of information, processes, or finance.

- A project manager is responsible for managing several concurrent and interconnected projects. One does not use the models directly but needs to understand the concepts behind them.

- A software vendor is a commercial company with developers. One needs the information to verify that the developed software is compliant with the necessary business functionality.

In each practical environment we encountered, we made improvements to the presentation of models. Next, we improved the way of working to create those models in order to make it more complete, based on the information acquired with the previous iteration(s) of the model. Consequently, every iteration has influenced the creation of the next ones. The stakeholder types did not change between the iterations. When, upon validation, the results of the models were not satisfactory, we specified the changes for the next iteration.

6.2 Practice: Medical Whole Sale (MWS)

Case A is a medical wholesale organisation that only needed a construction model and a fact model of their organisation to be aware of the organisation and communication structure. We modelled around 49 TKs, 38 EARs, 12 CARs, 17 ETs, 4 ARSs, 0 WISs. We modelled their organisational structure in DEMO and combined it with the landscape in ArchiMate. Although only Organisation Construction Diagram (OCD) have been made, the combination between the OCD of DEMO and the Application Layer Diagram (ALD) of ArchiMate have been very useful, therefore confirming the need of req. 3 on page 130. In this case, the connections between DEMO and other notations have been identified.

6.2.1 Introduction Case MWS

The company in iteration A was in the process of rebuilding its automation through the implementation of a new software system. At the start of the modelling, a Design and Engineering Methodology for Organisations (DEMO) Construction Model (CM) had already been built as a representation of the processes throughout the project. However, the project team responsible for software implementation only focused on functionality. Moreover, the construction focus of DEMO was not understood by some of the involved stakeholders. We stepped into the project when it had already been in progress for 18 months. Subsequently, in the next year of the project, we tried to embed the constructional philosophy of DEMO into the project, but we failed to reach the developers and project leaders while other stakeholders got hold of the concept. Changing the paradigm of a solution for the various stakeholders is very difficult after the start of a project.
The working methods of the implementation partner of their software system had already been in place for many years. They included an agile paradigm from a functional point of view. Moreover, the company assumed that the functional behaviour of the software system was 80% correct. Therefore, the only remaining task for the developer was to adjust the functional behaviour to the findings of the user during testing. Although this

was not an engineering approach, it could eventually lead to the final product. Aside from the developers, the program manager was too busy to embrace the DEMO methodology for the project as there were too many details to take care of. As is often the case[100], the project eventually grew too big when more people came in to do the hands-on jobs without a profound backbone structure.

6.2.2 Construction of Case MWS

Since the construction philosophy did not stick, We had changed the Transaction Kind naming to accommodate more functional names. In this way, the CM became a process-oriented model that was understood by many people, including the operational employees. We created the CM based on user input. The model was described at the ontology level, but it was insufficiently detailed for use during implementation. Therefore, in the last six months of the project, we extended the CM with the Process Model (PM) and with Transaction Pattern Diagrams (TPDs). In two-hour interviews with two to three employees, we captured the information about every step in the process and the connections between these steps. Besides, we followed the complete transaction pattern to add all practical situations that could occur.
At the beginning of the testing stage of the project, We issued a request for checking the completeness of the test scripts. We discovered that the number of tests was not sufficient to guarantee the robustness of the software system. With a model of over 15 transaction kinds, 5 business rules, and 20 steps per transaction, you would expect around $15 * 5 * 20 = 1500$ test scripts. With only 140 tests for user stories that would match at most 2 process steps per script and overlapping business rules, we roughly estimated that at least 300 scripts were missing. We used DEMO, and more specifically the TPDs, to match all test scripts with the Construction Model. Also, we added over 170 test scripts to the existing scripts that we found in the TPDs. Unfortunately, there was insufficient time to complete the model using this approach, so we were only able to share partial results on this subject (see fig. 6.1).

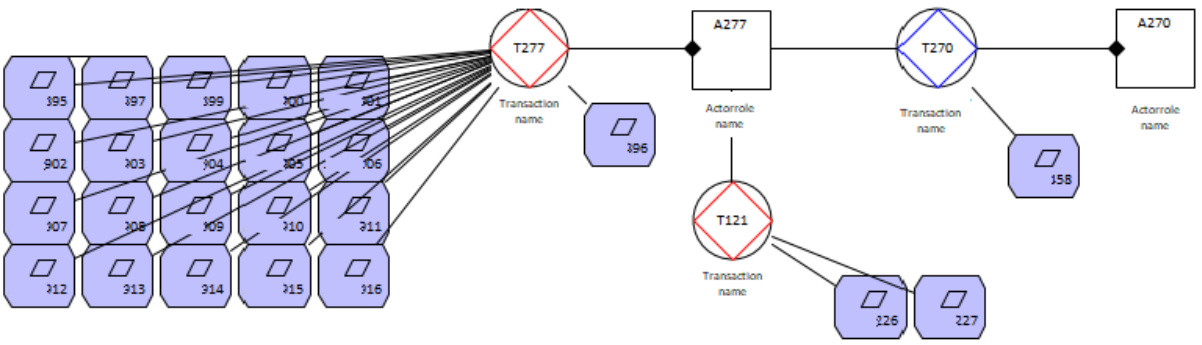


Figure 6.1: Iteration A: OCD with test scripts

6.2.3 Operationalization Case MWS

We presented the CM and the TPDs to the users that were part of the operational business. With the additional explanation, these stakeholders could understand that the diagrams were a representation of their process. However, in the end, we felt that they did not grasp the model well enough to increase their understanding of the process. During a presentation for combined groups, the CM did trigger interdepartmental conversations about processes and about how information was or was not communicated between those groups. Moreover, the responsibilities of the department were easily revealed and understood by the stakeholders.
Discussing the models with the testers was more difficult. Testers have a functional view of user stories, and their focus is not per se on all the steps that can go wrong. Especially when time pressure increases, their willingness to co-operate with new methods decreases. Therefore, although we have created many new test situations, we needed more of them to complete the model and ensure the success of the system. The developers also did not fully understand the TPDs with test scripts (see fig. 6.2).

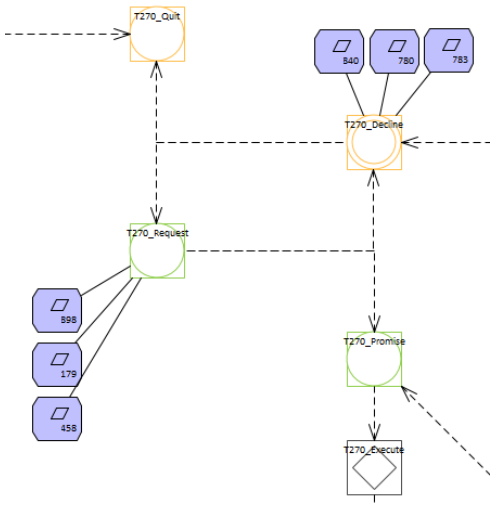


Figure 6.2: Iteration A: TPD with test scripts

We want to emphasise that no other models were present at the beginning of this project in order to be complete. During the project, these models were introduced, causing similar issues as the DEMO models. Therefore, it is not clear from this iteration alone whether the representations, the models, or the lack of understanding of the models is the cause of failing to properly apply DEMO in this project.

6.2.4 Conclusion and Further Research Case MWS

Concluding, our findings in this iteration were:

- Operational employees and developers do not understand the default representations of the CM and PM.

- Using process names for Transaction Kinds can help users understand the process. Nevertheless, this subverts the CM because the transaction kinds will be regarded as simple processes or even simple process steps.
- The models must be constructional to be useful in future tasks.
- The models must be sufficiently detailed for the goal of usage. Although this sounds trivial, methods do not specify the level of detail needed.
- The project must embrace the DEMO methodology to release its full potential. Not only do people have to start with DEMO, but they also have to maintain this view throughout the entire project to profit from the benefits of the methodology.
- In the end, the understanding of the software system implementation was dogged by many issues which might have been prevented if we could used DEMO to its full potential.

6.3 Practice: Real Estate (RE)

Part of this chapter has been previously published in the paper for the EEWC 2019 conference[2]. Case B is a property management company that needed its processes and data modelled to be able to choose the right automation for their business. We modelled around 222 TKs, 191 EARs, 36 CARs, 149 ETs, 0 ARSs, 9 WISs. The complexity, in this case, was in the implementation. The OCD and OFD were the initial implementation versions of both the landscape that we modelled in ArchiMate, as well as the application mapping. The OFD was used for the base of the configuration of the domain application. The combination of the OFD and implemented entities is a concept that fulfils some demands and can be expanded. The security model that was needed for the application could neither be registered in DEMO nor in ArchiMate and needed further attention.

6.3.1 Introduction Case RE

The company in iteration B wanted to start a new software implementation for its property management system. The choice of an implementation partner for the system had to be tendered following government regulations. In preparation for this tender registration, we produced the CM and the Fact Model (FM) of the relevant parts of the organisation. They were included in the tender request. Fortunately, the Object Fact Diagrams (OFDs) in the FM were understood, but no participant could fully understand the impact of the OCDs in the CM. This resulted in a presumptuous attitude towards the gathered business information because only the models they were used to were understandable to them.

6.3.2 Construction of Case RE

In this iteration, we used the CM to describe the processes of the company; the FM to describe the data structure, requirements, and responsibilities. The OFD was extended

with the implementation data model of the implemented software. By combining the two models, the theoretical model and the implementation model could be compared (see fig. 6.3) and the functionality of the implementation evaluated for the proper data structure. Also, an Actor role Function Diagram (AFD) was produced to show the relationship between actor roles and organisational functions, which revealed the mismatch in role and function definitions.

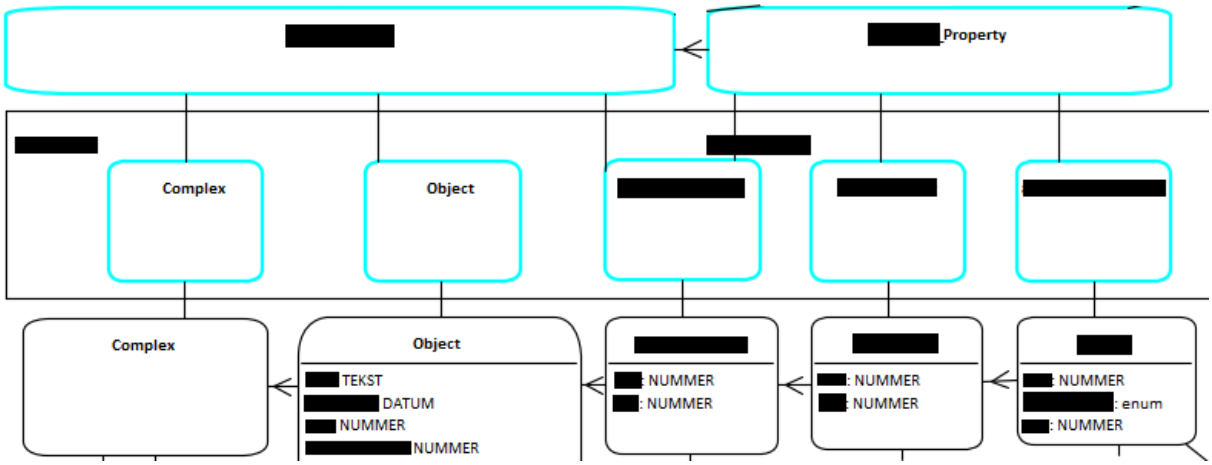


Figure 6.3: Iteration B: OFD with implementation (Dutch model)

Unfortunately, neither the software solution nor the software partner was able to produce a complete data model of their software upfront. Therefore, the model comparison could not take place before contracting the supplier. During the project, we found out that the implementation partner was not able to understand our abstract models. Therefore, the creation of the final model was a struggle. As a consequence, the data model was adapted several times during the implementation process to be able to comply with the requirements.

The execution of DEMO models [39] can be helpful to formalise the workflow within an organisation. The software solution of the chosen supplier had a workflow engine that was based on a Petri net implementation. The DEMO process model can be transformed into a Petri net implementation[101]. Therefore, we assumed that it should be possible to transform the DEMO process model into the software. Unfortunately, the vendor only had a proprietary interface to the workflow structure. Therefore, this option could not be tested even though it might be a very solid way to start workflow implementation.

In this project, we made the first trial with the new concept of security roles. A security role is the implementation equivalent of the authorization part of the actor role in the context of Identity Access Management (IAM). It determines the read- or write-relation with implementation entity kinds. The anticipated usage of security roles is to enable IAM from a model point of view and to visualise this information (see fig. 6.4). To our knowledge, no other modelling tool can provide this feature. Because this SRD was made experimentally, no formal definition was made upfront. Next to security roles, security functions have been defined (see fig. 6.4 left SF08). These security functions are the em-

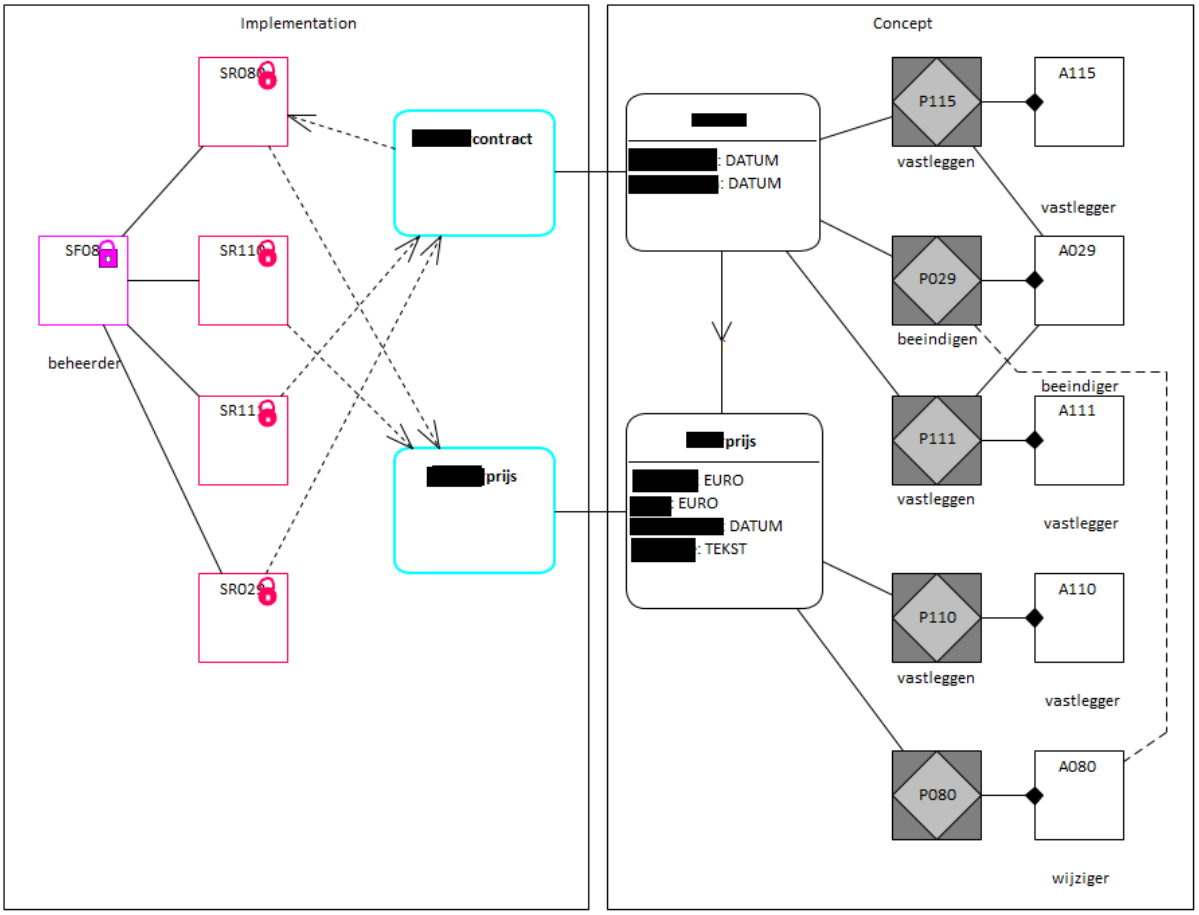


Figure 6.4: Iteration B: Security Role Diagram (SRD) example (Dutch model)

bodiment of groups of functionality usually described as security groups in Windows. The N:M relation between security roles and security groups makes the information complex and complete. Actors can be assigned to security functions to grant the rights to access data and, consequently, processes.
The interviews we held in this company were recorded. By using these recordings, we could capture more information from a single session. This information was used as input for a second session with the interviewee in which we obtained feedback on the model to help understand their own business from the produced model. This approach enabled us not only to achieve model completeness but also to determine the level of understanding of the model by the interviewee.

6.3.3 Operationalization Case RE

The CM of the organisation was presented in a layered set of OCDs and Transaction Product Tables (TPTs). From the main diagram, the detailed diagrams were accessible in an interactive visualisation. This first top-down approach was not without challenges: we had to make the CM understandable to the employees. The drill-down visualisation

helped to determine the context, but it was not sufficient enough for the stakeholders to understand the construction perspective on their processes.
The CM and FM were divided into departments to get a grip on specialised parts of the model. This proved to increase the enhanced focus on partial models during our interviews and feedback sessions. The partial model was particularly helpful to the stakeholders during the feedback sessions. These partial models concerned their own piece of the enterprise and were comprehensible for these stakeholders.

6.3.4 Conclusion and Further Research Case RE

The findings in this iteration are:

- The software product must be accessible to reverse engineer the product to data and process models for it to be verified. The software vendor must be flexible towards changes in the new implementation method.

6.4 Practice: Online Retail (OR)

Case C is a small call centre that needed to choose a process and data matching application. We modelled the organisation and their landscape using DEMO (OCD and OFD) in combination with ArchiMate. We modelled around 58 TKs, 21 EARs, 19 CARs, 32 ETs, 0 ARSs, 55 WISs (see fig. 6.5). By reverse-engineering the software towards DEMO models we found an 80% match in the process and data model. This was complete enough to buy the software licenses. Matching implementation and the organisation model is a part that is not completely covered by the tool in the used version. The metamodel lacked some connections between the various entity types.

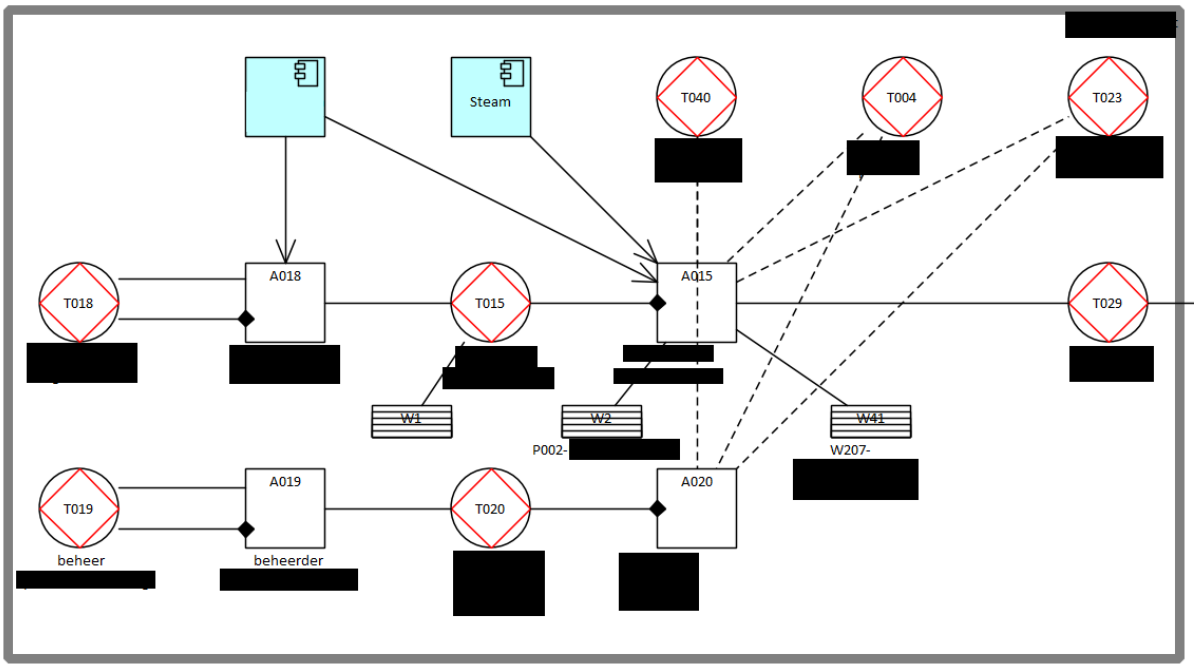


Figure 6.5: Partial CM/OCD case C

6.4.1 Introduction Case OR

The Company in iteration C, needed to buy (not to build) a new software program to support their business. Therefore, we first analysed its business needs.
We built the DEMO model using input from interviews. The interviews were mainly aimed at obtaining transaction kinds, actor roles, and entity types. The feedback on the model was obtained directly by creating the OCD and TPT live with the interviewees. The data model, or FM, was created from other information we obtained during the interview. This information was more complex. The main issue in this iteration was the terminology. When the terminology is complex, and the definitions are not clear or not present, it is difficult to create a correct data model. Therefore, the model was created in a few iterations. The final model was understood by the operational employees as well as by the directors.

6.4.2 Operationalization Case OR

The FM was presented to the software vendor in the standard OFD format and was understood according to the feedback. However, the vendor only had a data model for the base implementation, including all entity types of the system. This made an explanation of the model more difficult. Next, the data model was used to extend the model from a functional perspective, not from a constructional perspective. Moreover, to obtain the relevant data model was difficult for the software vendor. Also, the implementation model was too different from a logical data model, which made a direct comparison more difficult. Finally, the logical OFD representation did not fully match the implementation model. After refining the differences, we came to an 80% match on the data model, which indicated that, after detailed analyses, the program could store the information that the business needed.
The CM, used as a representation of the process, was not understood by the software vendor. The software vendor had only implemented functionality, divided into software modules, that described the activities that were performed on the invocation of a menu option. This functional implementation versus abstract construction mismatch made the CM less useful for explanation to the software vendor. In conclusion, our findings comply with those reported in [102].

6.4.3 Conclusion and Further Research Case OR

The findings in this iteration are:

- Data models rely heavily on definitions. Before creating the data model, a list of definitions has to be made. This action should be added to the methodology.
- The OFD can be useful for comparing the match between a theoretical data model and an implementation data model when visualised in the same format.
- The OCD is generally not useful to match implemented functionality with organisation building blocks.

6.5 Practice: Retail Whole Sale (RWS)

Case D is a logistic wholesale company that was seeking business optimisation. We mainly used the OCD to gather all products and processes within their organisation. We modelled around 200 TKs, 168 EARs, 81 CARs, 107 ETs, 0 ARS, 3 WISs. With the help of the integration of the Disco process mining tool, we gathered information at the main TK and related that information to cash flow, processing time and material flows. The tool supported this modelling by combining process mining information, ArchiMate landscape information and transaction and role information. Business optimisation was found in the analysis of the connections between the different departments [43].

6.5.1 Introduction Case RWS

The company in iteration D wanted to have a scan of the organisation to assess its efficiency. Therefore, we did a full scan of the business processes and the validation of the business data requirements. During this process, we concluded that we needed a new approach for this iteration. The Organisational Essence Revealing (OER) method of Design and Engineering Methodology for Organisations (DEMO) shows how to extract the essence from the implementation by communicating with the people involved. This is a good method to obtain the aspect models in a DEMO analysis, including the original, infological, and datalogical layers. This also allows for re-modelling the green and blue layers when needed. The OER method does intentionally not help with extracting the implementation components themselves (e.g. application components, interfaces). Therefore, we added more models and methods to capture this information.

6.5.2 Construction of Case RWS

In company D, we created an almost-transcription (called 'statement' that contains the essence of what is said, but not the full literal text) of a recorded interview. This statement was then used as the basis of our information analysis. The analysis was done using the following interconnected diagrams: the Organisation Construction Diagram and Transaction Product Table for transaction information, the Application Layer Diagram (ALD) from ArchiMate for application information and interface information, and the Object Fact Diagram (OFD) for the data model.
The Construction Model (CM) was created using all Transaction Kinds (TKs) and actor roles revealed from the written statement. Therefore, we recorded every relation between the identified TKs and the statement text that originated that TK. These comments were used afterwards to report our findings within the organisation. In the end, retrieving all comments in the model took a lot of effort.
The multidisciplinary team in this project worked on the same repository, in the same tool, leading to a consistent model. After adding the separate viewpoints, the connections between the viewpoints had to be added. This also took a lot of effort because all parties had to be involved.

6.5.3 Operationalization Case RWS

The main stakeholders for this iteration were an operational employee, a director and a C-level director. The CM was presented and was comprehensive to the operational employee. The information was clear enough to retrieve unmatched responsibilities that could be added to the model using the Elementary Construction Flaw (ECF) notation [103]. The CM presentation on a company and department level was also good for the director and gave the director insight into ECF issues and interdepartmental communication. Also, wrongly allocated responsibilities were identified and communicated. For a C-level director, the CM was not suitable. The mismatch between the knowledge and abstraction level of the stakeholder and the modeller made communication difficult and reduced the perceived benefits of the modelling effort. We successfully added the Flowchart (FC) viewpoint to reflect the initial knowledge and abstraction level of this stakeholder.
This iteration was also the first to use poster format diagrams. We used two types of posters. First, we created a poster about the relationships between the diagrams of several modelling standards. This poster showed that the Organisation Construction Diagram (OCD) has TKs which are decomposed in a Transaction Pattern Diagram (TPD) where each Transaction Process Step Kind is connected to an Action Rules Diagram (ARD) to reveal final conditions and attribute type usages. Secondly, we created a CM poster that had the complete CM. This poster provided insight into the complexity of the organisation. The diagrams poster (metamodel) was very helpful in explaining the DEMO methodology and the relationships between all diagrams to C-level stakeholders. It showed the possibilities and complexity of the modelling environment. The CM poster itself (model) was less helpful for the C-level stakeholders. However, it helped explain to directors the relation between the organisation parts which had been revealed and also revealed the important issues at stake.
The drill-down reporting, as already initiated in iteration B, section 6.2.3, was further developed. The mental division between functional processes, constructional models, data or application models and value streams and measurements were added to the drill-down order. Aspects within a diagram were coloured. Now a multi-dimensional set of diagrams can represent each view of a stakeholder. This model construct, despite adding complexity, has a positive impact on the Return On Modeling Effort (ROME)[104]. The modelling effort for creating all elements from different viewpoints is relatively low when starting with the creation of the model. The level of understanding of the stakeholders is worth far more than this initial investment. It might be that some parts of the model will never be seen by some stakeholders, but the total model is comprehensible to the entire target audience.

6.5.4 Conclusion and Further Research Case RWS

The findings in this iteration are:

- Each stakeholder needs his/her representation. Although a certain level of abstraction might be good for engineering purposes, it can also hinder the stakeholder's understanding of the complete model.

- The order of building the model may be more important than the DEMO course envisages, and re-evaluating information during the modelling is inefficient.
- Finding information and highlighting information in an existing DEMO model is not easy. Marking the highlights when the model elements are created is easier.

6.6 Practice: Whole Sale Administration (WSA)

Case E is a logistic wholesale company that lacked insight into their invoicing system. We modelled around 23 TKs, 20 EARs, 10 CARs, 12 ETs, 0 ARSs, 0 WISs. This analysis using just the OCD resulted in the insight that the communication was scattered around the organisation, and the accompanying responsibilities were not well defined. Implementation of a single actor role was among several subjects. The presentation of this information needed some improvement in the tool.

6.6.1 Introduction Case WSA

For the company in iteration E, we built a model of a single operational process. While this process was relatively small, the modelling depth was very high. This modelling aimed at finding optimisation possibilities.

6.6.2 Construction of Case WSA

Based on the changes in the previous iterations, our approach to creating the model was different. First, instead of interviewing all stakeholders upfront, the interviews per stakeholder were directly modelled in thirteen interconnected diagrams before the next interview started. These diagrams are Business Layer Diagram (BLD), ALD, Technology Layer Diagram (TLD), FC, OFD, OCD, Process Structure Diagram (PSD), Organisation Hierarchy Diagram (OHD), Actor role Function Diagram (AFD), Actor role Competence Diagram (ACD) and some combinations of the mentioned diagrams. The BLD, ALD and TLD are existing diagrams of ArchiMate, but are now added as default diagrams in our business model analysis; therefore, expanding the viewpoints of the model. FC is also a standard notation. The OHD is an extension to the DEMO diagrams showing the hierarchical ordering of Composite Actor Roles (CARs).
Subsequently, the creation of the integrated model was performed using a breadth-first approach instead of a depth-first approach. This made the model more consistent. The depth-first approach, like the OER method, focused on building the aspect models as a whole. It tries to reveal the transactions and actor roles and their relations. This results in an OCD and TPT representation. From this point, the OER method advises the modeller to go through the Process Model (PM), Action Model (AM), and Fact Model (FM) aspect models to complete the DEMO model. This might lead to some iterations due to missing information in one or more aspect models. In the breadth-first approach, we analyse each sentence as they are uttered (or, in our case, as present in recordings of interviews). After writing down the sentence in an almost-transcript, the sentence then is specialised

into all model element types (i.e TK, AR, Application Component etc.) that we want to model. Each element is entered into the model. Moreover, all elements are connected to the existing part of the model upon entry thus extending the model in each step. Using this approach not only saves time compared to the depth-first approach because the recording is listened to only once, it also improves the model on each iteration. All model information is complete at each moment in the modelling process.
It has been noted that people tend to think in flows. In the applied approach, which we called the eXtended Organisational Essence and Revealing (XOER) method, after notation of the sentence in a statement, we reveal the functional models (e.g. flowcharts, Business Process Model and Notation (BPMN)) and move from these functional models towards abstract, constructional models where we write down every step in the analysis. Moreover, this notation supports the thinking process of the analyst and above all, the thinking pattern of the user of the model.

6.6.3 Operationalization Case WSA

Due to the multiple models at various abstraction levels, it appeared to be possible to accommodate all stakeholders. The Flowchart (FC) accommodated the higher and lower management levels, whereas the OCD and TPD of the respective aspect models helped to explain and maintain the integrity of the model and the processes. Also, the data model helped us to reveal missing processes and responsibilities in the master data management process. Next to the diagrams that are used to transfer information, the way of presenting these diagrams to the stakeholders also matters. In this iteration, we used three different methods to share information. First, we shared the raw information which were contained in the interviews. The interviews were written as a continuous story, which, therefore, fully represents the information as it was uttered by the interviewee. Next, a document was produced with the diagrams, diagram descriptions, elements and element description in a logical implementation-abstraction order. This guided the reader from implementation towards the abstract blueprint of the organisation. All implementation processes and construction diagrams were categorised by functional departments. Finally, a clickable web iteration of the model was produced, allowing the user of the model to drill down to the level needed for understanding the model. The completeness of the models and the multiple media used to communicate the diagram increased the understanding of the model. Further research is needed to rule out other variables that could potentially influence the stakeholders.

6.6.4 Conclusion and Further Research Case WSA

The findings in this iteration are:

- Multiple viewpoints accommodate multiple stakeholders with the same information.
- The modelling effort of multiple diagrams is not significantly higher than modelling a single aspect. When more diagrams and aspects are created, they support the thinking process about the modelling steps. Therefore, only the action needed to

create the elements and connections takes up more time. After creation, the extra modelling supports the reproduction of the thoughts of involved stakeholders.

- The modelling effectiveness towards stakeholders of using multiple viewpoints during modelling is higher when compared to only modelling the two or three DEMO aspect models. When multiple stakeholders receive modelling support, they can see the connection between their thoughts and the DEMO models.
- The modelling integrity and completeness of the models are higher when all the models are used at the same time. When using a depth-first approach, one might miss model elements when revisiting a certain process than when handling the entire issue at once.
- Using the breadth-first approach gives modelling teams a better chance to work together on the complete model.

6.7 Extending OER

While the Organisational Essence Revealing (OER) method is helpful to reveal the essence of the organisation, we need to extend the method to reveal more aspects of the current implementation in a single pass. The current OER method [9, Ch.12] states that (1) the structure or essence is already present, and (2) the most valuable parts are in the people. The way to reveal the organisation is to talk with people and/or interpret the documents. In the last iteration, we have practised this extended method and found promising results. The extension of OER to multiple models modelling allows for addressing more types of stakeholders with minimal extra effort.
While the OER method explains to abstract from realisation and implementation the XOER method does abstract but splits the realisation and implementation from the essence of the organisation. This splitting of the organisation divides the implementation of procedures and automation and administrative procedures from the essence revealing two associated worlds. After all, the implementation world is supposed to support the essential world on various aspects.
The separation of concerns not only applies to the constructional and functional parts of the organisation, it also applies to the layers, though they are less precise, as defined in ArchiMate (e.g./ Business, Application, Technology, etc.) and other notations. Taking into account all these concerns that an organisation wants to reveal, the model can be created from the same information source.
The use of abstraction applies to the O, I, and D organisation as well as the departmental configuration and the application components involved. Revealing the elementary actor roles most often also reveal the functionary types (or job roles) that we have stored in Actor types in our iterations. Using the already present information in the organisation about this functional business layer can result in interesting insights about job roles that have too many responsibilities.
Devising the proper concepts exists on all levels of abstraction. This investment is good for the organisation at all levels. Finding the right names for products that work for

the processes and the facts is one thing, finding names that also work in a glossary or application environment is another. Concepts work at various departments in the organisation and need to have a definition in terms of the culture of that department.
Of course we should apply verification by implementation at all parts of the model. We choose to use the implementation in the model rather than only abstracting from it. Now we automatically have a partial intrinsic verification that appeals to the organisation because they know these concepts. Thereby we extend the verification from the usual suspects of data verification to more concepts of the metamodel.
The last step of validation is important for all concepts in the overall model. For this validation we not only need the people that fulfil the identified actor roles, but also the people that support the core processes. Only when the full organisation validates the model one can make assumptions about the consequences of changes in that organisation. The nature of DEMO allows for a modular approach in order to reveal the essence of the organisation. This does not have to change when we add implementation concepts to the model.

Iteration	0	A	B	C	D	E
Viewpoints (added)	OCD, PSD, OFD, TPT	TPD, Testscript	AFD, SRD	Definitions	ALD, FC, ARD	BLD, TLD, OHD
Process	OER	interactive		interview	written interview	XOER

Table 6.1: Added aspects in iterations

This multi-method approach leads to the question of how to integrate these viewpoints into a (single) metamodel. As all viewpoints originate from the same information these model components are connected. Though not all viewpoints are found in every iteration, all viewpoints seem to be relevant in a certain context. Therefore, we can conclude that the metamodel must be extendable for new elements, relations and viewpoints.

Stakeholders	Viewpoints and conventions output	Modelling input
developer	SRD, definitions, ALD, TLD	
operational	functional names, definitions, BLD	interview
director	definitions , OCD, BLD	interview
C-level	functions, FC, High-level OCD	
manager	definitions, OCD, FC, BLD	interview
vendor	methodology, OFD, OCD, ALD, TLD	data model, process model

Table 6.2: Stakeholders and the found representations

This approach eXtended Organisational Essence and Revealing (XOER) is intended to be part of a full methodology in our future research.

6.8 Gamification

As mentioned before in our discussion regarding the meaning of enterprise-grade in section 2.8, since enterprise engineering typically involves many different aspects, as well as many different stakeholders [36, 44, 37], it is important for modelling tools to provide different visualisations, in order to make information comprehensive and understandable for all involved stakeholders in all layers of an organisation. For instance, the Organisation Construction Diagram (OCD) does not work for C-level stakeholders [4]. C-level prefers high-level blocks and arrow representations of processes, such as provided by the Action Rules Specification (ARS) level diagrams. Similarly, towards C-level management, the Process Structure Diagram (PSD) also does not work well when dealing with complex processes. Finally, the Object Fact Diagram (OFD), when completely drawn, is also uneasy to read for a lot of people. Based on the cases we found that dividing it into several partial models helps to make aspect models clear to stakeholders.
As such, in line with [44], it is necessary to distinguish between different visualisation strategies for different stakeholder groups and for different purposes. For instance, visualisation can also involve the creation of animation of a process, or even gamification. Enabling stakeholders to simulate, or even "play" with these visualisations, their processes allows them to see the modelled processes as they are running. It also combines the view on processes with the production facts that are created in a transaction.
Our first attempt with gamification of Design and Engineering Methodology for Organisations (DEMO) models (see fig. 6.6) visualises the OCD in a 3D setting where the actor roles are visualised as people doing tasks behind their desks. This visualisation is supported by a multi-user environment that can play one or more roles within the gamification scope. This setting allows for multiple roles played by a single player, therefore, restricting the execution of multiple roles with only one actor to fulfil all labour.

Figure 6.6: Simulation of a DEMO model

These experiences require new visualisations to be created, evaluated and improved. When new insights appear, first, the metamodel needs to be extended with all new concepts that

then emerge in the visualisation and implementation activities.

6.9 Findings

Practitioners daily experience the struggle to (1) explain the models to customers, to make them (2) understand the diagrams and tables, and to (3) explain the theory of DEMO. While some customers simply refuse to understand, others fail to grasp the abstraction level that is inherent to the methodology. In making DEMO more accessible for relevant stakeholders, various visualising concepts have been used to reduce the complexity, to show the integrity of the model, and enhance its comprehensiveness. However, most visualisations went beyond the understanding of some of the stakeholders.
Renovating one's bathroom might serve as a useful example here. When the construction worker explains where the pipes in the walls should be put, one will probably understand the construction. This expert is needed to create this type of model. Management in the building company understands the function of construction models: they enable the construction to run on a daily basis and assure interconnections and smooth operations. To return to our example, project leaders do not need to know where the pipes should be. They only need to know the connections between the associated workers who have to finish the work on time and give them relevant information in the right order. This brings us to upper management. They do not need to know about the construction model; they are interested in a more efficient business operation seen from a functional perspective. Or, in construction terms, they want to know whether it is possible to build cheaper, faster, and more efficiently. The information that enables these insights starts at the construction of the organisation and needs translation on the way up. It also needs a representation that relates to the construction model and its functionality.
The main research question that we have investigated is, whether the aspect model visualisations of DEMO contain the right properties for explanation to the stakeholders. This paper shares our findings in this search for modelling the construction of the organisation in a way understandable for relevant and associated stakeholders within all layers of the department or organisation (depending on the scope of interest) as well as explaining the 'way of working' of creating the model.

Part III

Results

Chapter 7

Results

7.1 Research Questions Revisited

The main research question throughout this research has been formulated in section 1.4.2:

> How can we formalise the DEMO metamodel in such a way that automated tools can assist modelling in DEMO and that DEMO models can be logically verified and exchanged for practical usage?

This question has been broken up in pieces and we will evaluate every piece of the question.

How can we formalise...	We have formalised the metamodel in chapter 4. We have used partial metamodels as described in section 2.6.
...the DEMO metamodel...	The partial metamodels have been build in sections sections 4.2 to 4.5 and we expanded these metamodels with practical model information in sections sections 4.7 and 4.8. Thereafter, we discussed the visualisation-metamodel in chapter 6.
...in such a way that automated tools can assist modelling in DEMO...	We discussed the implementation of a tool in chapter 5 and used this tool in real enterprises to model the organisation in chapter 6. For the tool we built memory-metamodels in appendix F.
...and that DEMO models can be logically verified...	We have defined the notion of verification rules in section 2.6. Subsequently, we used the notion to define the rules in section 4.1.2.
...and exchanged for practical usage	The exchange-metamodel has been built and has been used for a gamification model exchange in section 6.8.

Although the representation of the models is part of the DEMO Specification Language (DEMOSL) metamodel, we decided to define the representation of these models explicitly in a visualisation-metamodel in this thesis.
We have formulated our research objective in section 1.4.3.

> Our **research objective** is to
> formalise the metamodel of Design and Engineering Methodology for Organisations (DEMO)
> in such a way that
> DEMO models can be verified automatically and
> DEMO models can be exchanged.

To attain this result, we have to answer the following detailed research questions:

- How can we create the metamodel in such a way that we can implement it in a tool? We have created an ontological-metamodel section 4.1.1 with enough information to implement it in a memory-metamodel as outlined in appendix F. This memory-metamodel became the base of the tool's internal operation and made it possible to apply the verification rules. The tool visualisation was described in sections 5.2.4

and 5.2.6 and has been connected to this memory-metamodel using add-in functionality to provide the model elements and connections from a visualisation to internal representations.
The AM of DEMOSL was not complete enough, but has improved a lot compared to the original version. These are not only elements that could be missing in the tool and the metamodel, but often challenge the theory as well because the obvious parts of DEMO can be modelled with more ease than before.

- How can we verify a DEMO model in such a way that it is beneficial, and not a limiting factor, for modelling an organisation?
  In Design Science Research (DSR) terminology, the metamodel that was created from this specification was the first iteration of the DEMOSL artefact. Using all models from the DEMO book [25], and associated course material, we created a second iteration.
  Based on the metamodel of the second iteration artefact, a tool was created [105], which has been used in practice to model organisations. In doing so, all missing modelling elements have been added to the model and to the tool and used in successive cases. After more than seven real live cases we are convinced that the CM, PM, and FM are fairly complete to hold all elements and property types needed for modelling DEMO.
  The implementation of the tool (section 5.2) has been used in a number of real case implementations as described in chapter 6. These implementations resulted in the preliminary extension section 6.7 of the OER method of the DEMO methodology.
  A major challenge in DEMO visualisation remains the potential variety of stakeholders [44]. The more types of stakeholders the larger the number of viewpoints might be needed. To aid in bridging the gap between stakeholders in practice, we added a functional concept to the construction model. This is the functional value that the organisation gives to the construction model in its organisation. This one-on-one functional translation is the first step to connect the functional business and construction domains of DEMO.
- How can we use the DEMO model and the respective metamodel for automating the modelled organisation or parts of it?
  Building the tool, the exchange-metamodel and subsequent use of this tool shows that the use of the DEMO modelling methodology, supported by the tool, can provide verified models that can be presented to the enterprise and to validate the model of the organisation.

We conclude that all of the above mentioned research questions have been answered in this thesis. During the journey of our research, new questions have arisen and will be addressed in section 8.2.
Summarizing this revisit, we made adjustments in the DEMOSL to reflect practical needs in DEMO modelling and to enable automation of the modelling and validation of these models. Furthermore, we added steps in the methodology to cover a wider perspective of the modelling needs that do not necessary belong to the ontology domain, but surely are adjacent and often more visible to business stakeholders.
Exchanging DEMO models with other tools also needs other tools with the same

metamodel implementation. The only tool that has this metamodel in place is a gamification tool. This information exchange has been tested successfully (fig. 7.1).

Figure 7.1: Gamification Scene

7.2 Conclusions

In our effort to automate the verification and exchange DEMO metamodels, we have used DEMO, DEMOSL, and the tool support of Sparx Enterprise Architect (SEA) and we gathered experience in the use of these artefacts. We have created a metamodel that consists of the ontological-metamodel, visualisation-metamodel, data-exchange-metamodel, and visualisation-exchange-metamodel (fig. 4.3 on page page 92). Moreover, we have created a tool that consists of a toolbox, diagrams, shapes and shape scripts, ARS grammar, verification rules, the ontological-metamodel in XSD, memory metamodel, and visual memory metamodel. In the following sections, we will discuss conclusions of out research on the topics of DEMOSL, the metamodel, tooling, visualisation, and practical experiences.

7.2.1 DEMOSL

We started our research at the requirements in chapter 4 and compared the aspect metamodels to our automated support requirements in section 5.1.1. We can conclude that the original metamodels are incomplete, inconsistent, and not correct in their current state. The current state of this metamodel restricts the possibilities for automating DEMO model verification using these metamodels. Even though we have found several required improvements, DEMOSL has proven to be a solid base for our metamodel. In the context of a design science-based research effort, we refined the DEMOSL and extended it to serve as the foundation for an automated tool development. Furthermore, the extension of the DEMOSL makes it more comprehensible for involved stakeholders to create automation designs that support modelling in or with DEMO in practice. We find the extensions to DEMOSL to be relevant and useful.

7.2.2 Metamodel

Creating a metamodel for DEMO is possible from the foundation that DEMOSL has created. We were able to solve the omissions found as described in section 3.3 using the new ontological-metamodel and verification-metamodel. We thus resolved the need to improve the DEMOSL metamodel to be able to model all allowed DEMO models. We have now reached a stage where the metamodel is sufficient to automate the model's modelling and validation.

The Construction Model (CM), Process Model (PM), Fact Model (FM), and Action Model (AM) all have some defects in their metamodels as shown in sections 3.3.2 to 3.3.5. Therefore, we could not implement a correct representation of a DEMO model without making changes. Moreover, we built a verification-metamodel with verification rules to verify models. The verification-metamodel supersedes the methodologies current entity-type-based metamodel. DEMOSL needed a complete and restricted logical model of all allowed DEMO models. This verification-metamodel has references to entity types to address the attribute types and property types between all concepts. The verification-metamodel needs to be completed with all verification rules to create a full DEMO specification. We used it to help us find the verification rules for automating the logical model.

The ontological-metamodel has been modelled using the principles of Object Role Modeling (ORM), and has been expressed using the DEMO 3 notation. This modelling method is part of DEMO [25, ch5]. In practice, modelling the metamodel is done using the same method as the one used for modelling the data.

The eight improvements of section 4.2.1 have been added to the ontological-metamodel and the exchange-metamodel of the CM. These improvements make modelling an organisation easier and also enable the automatic verification of the model.

We have explained the three improvements on the PM's metamodel in section 4.3.1. Improvements in the ontological-metamodel enabled essential steps in storing the results of the analysis in the model.

The FM has been designed at an ontological level. This level was found to be too high for implementation, and we have detailed it further to make it implementable. We elaborated on the contents in section 4.4.1. The original ontological model has some concepts, expressed in entity types, that made it directly to the ontological-metamodel of the fact model. Other concepts have been removed or reduced because of the complexity reduction of the metamodel.

We found that the action model was insufficiently specified in order to accommodate specifications in real-life situations as shown in section 4.5.1. Therefore, we adapted the specification to make it consistent and applicable for more environments, e.g. automation and implementation. Moreover, we studied the usage of the DEMOBAKER [77] AM syntax to use it in the specification of the AM and found that this syntax is useful with some adaptations.

Summarised, we extended the DEMOSL to accommodate all practical found aspects of a DEMO model. The newly proposed DEMOSL can be used for automatic verification and for exchanging DEMO models in practice.

7.2.3 Tooling

After two years of testing (in real-world cases) and optimising the tool, we can conclude that, although not finished, the tool's current version is capable of storing all DEMO models and is sufficient for visualisation of the business processes. The ontological-metamodel implemented in the tool appears to be rich enough to store the information of the mentioned cases. The tool's base, SEA, is capable of supporting the modelling in DEMO. The resulting DEMO modelling tool has been used in practice and is available for researchers/lecturers in the academic world.
Even though the SEA platform allowed us to develop effective DEMO tool support, we also came across several limitations. For instance, SEA is unable to display tables. Nevertheless, with some effort, we managed to still visualise tables. A more fundamental limitation pertains to the representation of graphical images. The visualisation engine of SEA runs on a 100 × 100-pixel image resolution that will resize to the required dimensions. This resizing allows for most graphical visualisations (squares) but falls short for independent resizable shapes. This same graphical concept applies to connections between elements, which is also quite restricting within the SEA environment. Verification rules implemented in the add-on and the freedom within this SEA-add-on compensate for the internal tool framework's lack of possibilities.
Besides the discovered limitations of the chosen SEA tool framework, capturing the DEMOSL in SEA also revealed some missing definitions and inconsistent definitions in the DEMOSL during our research.

7.2.4 Visualisation

The big challenge in the visualisation of DEMO is on the variety of stakeholders confronted with this type of modelling. The more layers of an organisation are involved, the more types of stakeholders are addressed, e.g. employees, managers, C-level, the more significant number of viewpoints might be needed. In our practice experience, C-level management is the primary concern in finding the right set of viewpoints.
To bridge the gap in practice, because using schooling for educating the whole management level will take a generation, we have added a functional concept to the construction model. This concept consists of the functional value that the organisation gives to the construction model in its organisation. This one-on-one functional translation is the first step to connect the functional business and construction domains of DEMO.
We also found that DEMO can be used to explain the construction to lower management layers within the organisation and workers.

7.2.5 Experiences

In this section, we summarise some of our experiences in the development of tool support for DEMO. A more detailed account of these experiences is provided in [7]. This also involves the changes/refinement needed to the ontological-metamodel (resulting in the earlier discussed metamodel as shown in fig. 4.5), as well as the need to add a metamodel dealing with visualisation, and with the exchange of DEMO models, respectively.

A first interesting experience was the fact that, even though DEMO has a thorough theoretical basis, the original metamodel of DEMO was not specific and detailed enough to enable an immediate implementation. The latter can be explained by the fact that the book defining the DEMO method [25] was primarily written to teach learners (students and practitioners) to create DEMO models in accordance with the DEMO way of thinking, and draw (human to human) communicable DEMO models to reason about the organisation. As such, the book puts the priority on "doing", when introducing the different DEMO aspect model kinds. There was no need for a strict metamodel.
In line with this, the formalisation(s) provided in the original DEMO book [25] aimed to support didactic goals rather than the development of automated modelling tools. As such, it was never meant to provide a formalisation and metamodels that would enable the development of, and automated support for, the methodology. As discussed in [81], different meta-modelling and formalisation goals will / should also result in formalisations with different level(s) of detail/specificity.
As a result, to enable tool development, a more complete and detailed formalisation and metamodel was needed. This triggered the initial development, and further evolution, of the DEMOSL [1], the specification language for DEMO. A first validation of the metamodel of this specification language was reported in [2]. As part of the (partial) validation [2], all existing DEMO (example) aspect models (taken from the official course material and practical cases) were positioned within the specification language to see if they fitted.
As also discussed in section 7.2.2, beyond the ability to "capture" actual DEMO models within the specification language, more extensions of the metamodel were needed to enable verification of the models as well as the operational use of the method in practice [4].
During the implementation, and practical validation, it also turned out that, for practical purposes, some of the concepts could actually be removed from the metamodel. For instance, according to the theory, each Organisation Construction Diagram (OCD) corresponds to an explicitly defined Scope of Interest (SoI). In practice, however, the SoI always corresponds directly to a Composite Actor Role (CAR). As a consequence, in practice, the SoI concept is redundant, and therefore we decided to not have it included in the metamodel explicitly.
A further interesting finding was the fact that DEMO allows modellers to start from any of the four aspect models. When learning the method, one generally starts with a Construction Model (CM) and gradually works down to the Action Model (AM); see fig. 7.2. However, in practice, when interviewing domain experts in an organisation, these experts usually talk about the existing process and associated rules. In other words, starting with AM related information first. Additionally, the Fact Model (FM) information about entity types and attribute types is also provided relatively early when interviewing the domain experts.
The original DEMO metamodel did not allow for models to "grow" from the different aspect models, in the sense that the consistency rules would require the model to always be complete as a whole (so, including all aspect models). Therefore adjustments to the metamodel needed to be made to allow for such flexibility, while still enforcing (at the end of the modelling process) the overall consistency.

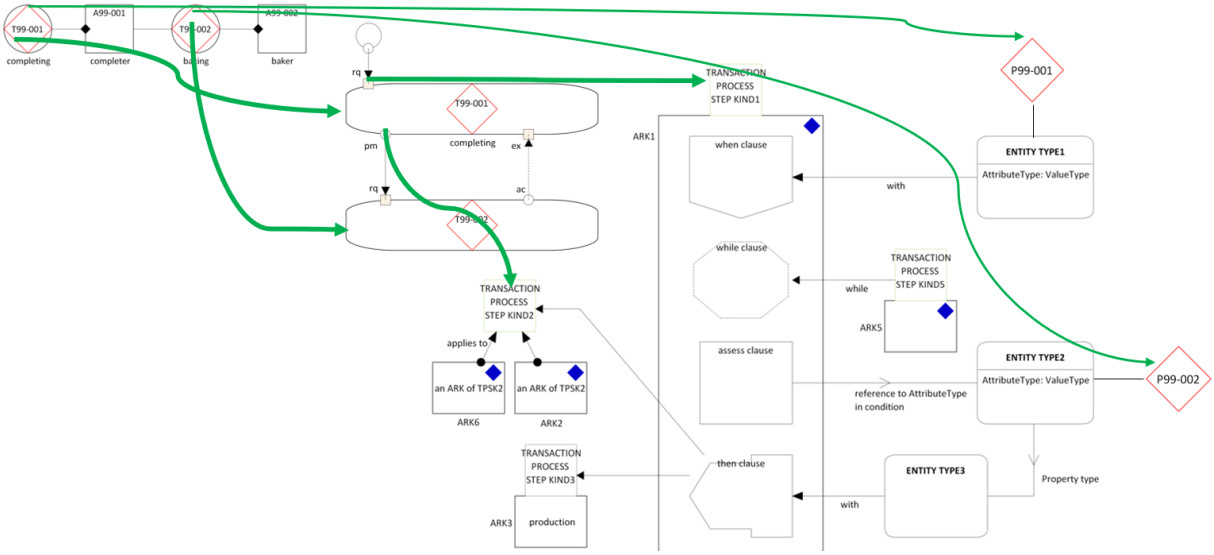


Figure 7.2: CM-AM flow

Furthermore, in practice, organisation models that resulted from the Organisational Essence Revealing (OER) analysis often raised questions regarding the *origins* of the included transaction kinds. More specifically, traceability from the elements in the organisation model to the original OER analysis was missing. To support this kind of cross-reference of the OER analysis, we have added the interview and interview line concept into the metamodel. By registering every aspect of the OER analysis as a connection from the interview line to the elements modelled from that line information we have created a traceable model from source to the final model. This interview notation technique is also illustrated in the NEN publication [19].

Another finding is that in practice, there was a need for DEMO models to be related to their existing or planned implementation. For instance, a "serving" connector was introduced that can be used to connect DEMO model elements to, e.g., application components in ArchiMate's Application layer. With this connector, one can, for instance, point to application components that implement the transaction kind or Transaction Process Step Kind (TPSK). Another example of the need to be able to include more of the implementation context involves the introduction of the actor (type) that aggregate actor roles. Such actors types correspond to job functions in the functional area of the model and, therefore, combine all competences of the elementary actor roles that are aggregated. This can be used for HR implementation information.

Finally, the action rule specification, as described in the DEMO method and associated specification language, consists of a semi formalised way to describe the communication actions that result in a decision on either actor role. The ontological metamodel of the action rule was composed of a single entity type that had its only connection to the TPSK it was based on. To start using the Action Rules Specification (Action Rules Specification (ARS)) in a more formal way, a more elaborate definition was needed. Therefore, based on earlier work [45], a grammar to represent action rules was created as well.

Next to such extensions (and simplifications) to the ontological-metamodel, we also found it necessary to add a visualisation, and an exchange-metamodel. Since the DEMOSL [1]

primarily focuses on consistency and completeness of DEMO models from a "content" perspective, it does not include any specifics about the actual representation of these models in terms of diagrams, tables and other possible visualisations. As such, the DEMOSL does not provide guidelines regarding the concrete syntax of models in terms of, e.g., shapes and icons to be used. When examining a corpus of models produced across different cases, we found various variations in drawings of elements that each could be interpreted as the convention of those elements. In moving towards (standardised) tool support, this had to be remedied in terms of an explicit metamodel of the allowed visualisations.
As an example, consider the diagram provided in fig. 7.3. In an OCD, the shape is normally drawn as a circle enclosing a diamond and this circle is stretched in the Process Structure Diagram (PSD) with the diamond displayed at a seemingly random position within this "stretched circle". In fig. 7.3, the name of the transaction appears below, above, or at the side-top of the transaction. Furthermore, the diamond-shape is positioned at a certain percentage from the left side, the stretching is a random length, while the swim lane usage is not consistent (i.e. 07 has the same initiator and executor but is visualised on top of two swim lanes).

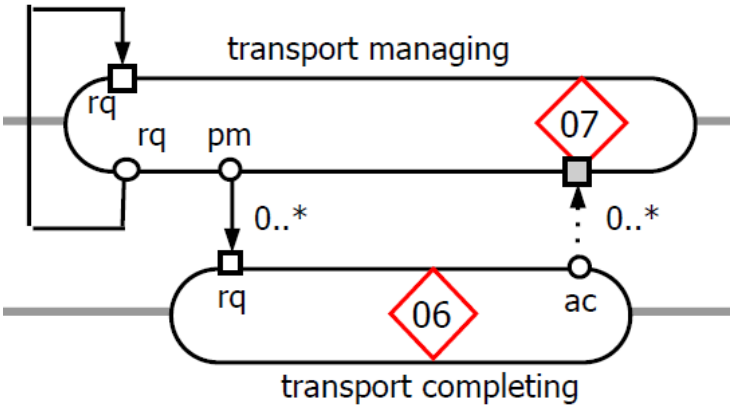


Figure 7.3: PSD specification example

This, finally, takes us to the need to exchange DEMO models between tools; see fig. 7.4 for an example. As reported in earlier work [105], a number of tools[1] can model (partial) DEMO models, while some of these even have a built-in engine to execute the actual models. It would be beneficial to be able to export and import DEMO models across tools because specific tools can have specific benefits when used in a certain organisational environment.
In addition, given the complementarity between modelling methods / languages, such as e3Value, BPMN, ArchiMate, and DEMO, it would be beneficial to also create conceptual bridges between the respective models. Experimental results regarding the potential benefits of this have been reported in e.g. [17, 63, 64, 51, 87, 88].
To enable (interoperable) export and import of DEMO models across different tools and engineers, an exchange-metamodel has been created that enables the exchange of both the

[1]Created by e.g. Bakker&Spees, Technia, Future Insight, and Formetis.

```
<?xml version="1.0" encoding="utf-8"?>
...
  <xs:element name="DEMOmodel">
    <xs:complexType>
      <xs:all>
        <xs:element name="TransactionKinds"
            minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="TransactionKind"
                type="TransactionKind"
                minOccurs="1" maxOccurs="unbounded">
...
  <xs:complexType name="TransactionKind">
    <xs:sequence>
      <xs:element name="Identification"
          type="TransactionKindId"></xs:element>
      <xs:element name="Name"
          type="TransactionKindName"/>
      <xs:element name="TransactionSort"
          type="TransactionSort" default="unknown"/>
    </xs:sequence>
    <xs:attribute name="Id"
          type="TransactionKindGuid" use="required"/>
  </xs:complexType>
...
  <xs:complexType
      name="OrganisationConstructionDiagram">
    <xs:all>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Formulation" type="xs:string"/>
      <xs:element name="TransactionKindElements"
          minOccurs="0" maxOccurs="1">
...
```

Figure 7.4: Part of exchange metamodel

actual model, including the different aspect models, as well as their visualisations. The exchange-metamodel enables translation and connection to various other languages and includes model extensions to store models in other modelling languages such as ArchiMate and Business Process Model and Notation (BPMN). The exchange-metamodel is based on XML Schema Definition (XSD), due to its common availability in modern-day development environments [3, 89].
A specific class of model exchanges needed in practice is the exchange between DEMO and 'Voorwaarden scheppen voor de Invoering van Standaardisatie ICT in de bouw' (VISI). As mentioned in section 2.1, VISI is an early descendant of DEMO which has now evolved into the ISO 29481-2:2012[2] (BIM related) standard for the construction sector. The exchange model between DEMO and VISI has been defined in the standard ISO 29481-2:2012.
In summary, the key experiences are:

- The enterprise ontology book [25] serves well for didactic purposes, but does not provide a formalisation of the method that can be used as a base for tool development.
- With regards to the actual DEMO method, the development of the tool, as well as the use of the tool in practical situation, resulted in:
  - The addition of extra concepts in the metamodels.

[2] `https://www.iso.org/obp/ui/#iso:std:iso:29481:-2:ed-1:v1:en`

  - The addition of support for interviews in the context of the Organisational Essence Revealing (OER) diagrams.

- With regards to the DEMOSL:
  - More support was needed to deal with flexibility of modelling processes.
  - The DEMOSL was validated and had to be detailed further, in order to enable automation.
  - The latter also included the formalisation of the ARS diagrams.
  - A visualisation metamodel needed to be added to cater for visualisations.
  - Similarly, an exchange metamodel needed to be added to support the exchange of models.

- Finally, to support the connection to other enterprise modelling approaches, relationships needed to be added between DEMO(SL) and some of these other languages.

Chapter 8

Follow-up

8.1 Recommendations

Findings and conclusions as elaborated on and described in this thesis point out improvements in DEMOSL that will subsequently improve the specification, methodology and practical use of DEMO. Various recommendations, ordered by subject, have been written for experts, researchers or practitioners of DEMO modelling.

8.1.1 DEMOSL

The readability of DEMOSL can be improved. The most ontological models can be complemented with practical models that allow users and tool manufacturers to start from the same and sufficient information level. Compact notation does not always explain the full description of what was meant in that notation. We discussed our DEMOSL findings in detail in chapter 3.

8.1.2 Metamodel

Tool manufacturers should at least work with the same standard exchange-metamodel. This exchange-metamodel will be publicly available and can be used broadly to have the same language in model exchange. Extensions on this standard exchange-metamodel can be requested. We recommend the enterprise that maintains the standard to be installed at the publishing date of this thesis. The standard exchange-metamodel could speed up the development a complete supporting environment for DEMO practitioners.
The verification rules are the basis for analysis and can help to improve the quality of models. In practice, not all rules are directly described in the DEMO theory. These verification rules need to be added to the methodology.

8.1.3 Tooling

It is unlikely that one can make a correct model of an entire organisation without using some sort of tooling. Tooling takes care of the internal verification of the model and can

provide a library of default solutions. All verification rules are applied automatically to the whole model, and the modeller can rely on the results in order to improve ones models. Imaging tools like PowerPoint and Visio (without adaptations) cannot contain the model that it visualises. Like all tools with a single visualisation, they are less likely to support modelling sufficiently. We recommend using appropriate tooling that supports multiple views on the same model and verifies the model to the internal modelling rules.

8.2 Future Research

Research triggers research. With this thesis written, the next topics will emerge. We will continue our research, and we do have an opinion on what one should research in the context of the subjects mentioned in this thesis.

8.2.1 DEMOSL

The metamodel described in this thesis allows for the extension into other information into the DEMOSL metamodel. This extension needs to be developed based on the line set out in this thesis.
In all aspect models there is room for improvement for practical use. While this research commenced, a new version of DEMO and DEMOSL was published. This new version consists of new insights, views, and reference models. We need more research on how to integrate the information in this thesis into the complete methodology. Conflicting ideas and practical use changes will have to be examined first in order to find restrictions in theoretical or practical usage. When these restrictions can be solved, one can combine all the information into a new version of DEMO and DEMOSL.
Since a new version[1] of DEMO has been published in 2020, we need to investigate the gap between the current state of the research on this topic and the new information available.

8.2.2 Metamodel

We could make a distinction between the concrete and abstract syntax of the metamodel. In the end, the only relevance is the usage of the metamodel. In future research, the intended use of the metamodel needs to be addressed first.
Therefore, part of the research should evaluate the amount of benefit by using Fact Based Modelling (FBM) methods.
We will further expand this metamodel to cover the whole of DEMO in order to be able to exchange all information currently used in DEMO models around the world.
Development on the side of the DEMO methodology needs support in the metamodels, information and automation.
As a next step, the metamodels will be extended even further to completely support DEMO 3 and the successor DEMO 4. We certainly need more research on the compliance

[1] `https://demo.nl/download/demo-specification-language-4-6-1/?wpdmdl=842`

of DEMO 4, and DEMOSL 4 to the current metamodels as the presented models in DEMOSL 4 already contain different entity types.
Furthermore, we expect that further extensions to the core metamodel will be needed in order to enable sensible connections to complementary modelling languages and methods. By now, research on the integration of DEMO and VISI has commenced. The progress of this part of the research so far has not been enough to add the results to this thesis yet. This transformation between VISI and DEMO is promising, and the combination and integration of metamodels can influence the usage of DEMO.

8.2.3 Tooling

Next to the existing SEA environment, we are also starting to perform research on several newly discovered tools/frameworks in order to investigate if these tools can also use the add-on that has been developed to extend the modelling capabilities towards DEMO.
The newer version of DEMO also involves new diagram types and tables, which will also have to be included in future versions of the tool. At first glance, some rules associated with the new diagrams and tables seem to complicate SEA modelling and might even go beyond its modelling capabilities.
We plan to elaborate on these insights in future work and will try to make additions to DEMOSL to compensate or complete the specification language.

8.2.4 Visualisation

In this thesis, we presented our first steps in the eXtended Organisational Essence and Revealing (XOER) method. Modelling all viewpoints of a model at the same time looks promising as it helps to model and to think. The upcoming iteration of the XOER method, which starts upon completion of this thesis, will help us make this method describable and educable. This upcoming iteration will be subject to an expert group to validate the metamodel and the used representations. We also expect that this validation will result in the need for new visualisations (including simulation and gamification) catering to the different stakeholders' needs. New research efforts have already been started to get more insight into the relationship between the complexity of the information and how we try to communicate this information in an understandable and comprehensible way to all relevant stakeholders when applying the DEMO methodology and tooling in practice.

8.2.5 Experiences

In future research, we expect to receive more feedback from practitioners based on the day-to-day use of DEMO. A lot of small and more significant models have been made in the last 20 years. Due to company legal and compliance restrictions, these models might not have been published or were not allowed to be published. Collecting this important information and feedback numbers might give insight into the usage, usefulness and problems or issues that other practitioners experience when using DEMO and DEMO tooling.

Bibliography

[1] J. L. G. Dietz and M. A. T. Mulder. *DEMO Specification Language 3.7*. 2017.

[2] M. A. T. Mulder. 'Validating the DEMO Specification Language'. In: *Enterprise Engineering Working Conference*. Springer, 2018, pp. 131–143.

[3] M. A. T. Mulder. 'Towards a Complete Metamodel for DEMO CM'. In: *OTM Confederated International Conferences' On the Move to Meaningful Internet Systems'*. Springer, 2018, pp. 97–106.

[4] M. A. T. Mulder. 'A design evaluation of an extension to the DEMO methodology'. In: *Advances in Enterprise Engineering X*. Advances in Enterprise Engineering XIII. Springer, 2019, pp. 55–65.

[5] Mark.A.T. Mulder and Henderik.A. Proper. 'Evolving the DEMO Specification Language'. In: *Enterprise Engineering Working Conference*. 2020.

[6] Mark.A.T. Mulder and Henderik.A. Proper. 'Towards Enterprise-Grade Tool Support for DEMO'. In: *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer. Nov. 2020, pp. 90–105. DOI: 10.1007/978-3-030-63479-7_7.

[7] Mark.A.T. Mulder and Henderik.A. Proper. 'On the Development of Enterprise-Grade Tool Support for the DEMO Method'. In: *CAiSE 2021*. 2021.

[8] Mark.A.T. Mulder and Henderik.A. Proper. 'On Enterprise-Grade Tool Support for DEMO'. In: *Journal Software and Systems Modeling*. 2021.

[9] J.L.G. Dietz and H.B.F. Mulder. *Enterprise Ontology: A Human-Centric Approach to Understanding the Essence of Organisation*. Springer Nature, 2020.

[10] M. Bjeković, H. A. Proper and J.-S. Sottet. 'Embracing Pragmatics'. In: *Conceptual Modeling - 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014. Proceedings*. Ed. by E. S. K. Yu et al. Vol. 8824. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2014, pp. 431–444. ISBN: 978-3-319-12205-2.

[11] H. A. Proper and G. Guizzardi. 'On Domain Conceptualization'. In: *Enterprise Engineering Working Conference*. Lecture Notes in Business Information Processing – Advances in Enterprise Engineering. accepted for publication. Springer, Heidelberg, Germany, 2020.

[12] Marc M. Lankhorst, Henderik Alex Proper and Henk Jonkers. 'The anatomy of the archimate language'. In: *International Journal of Information System Modeling and Design (IJISMD)* 1.1 (2010), pp. 1–32.

[13] V. E. van Reijswoud and J. L. G. Dietz. *DEMO Modelling Handbook*. 2nd. Vol. 1. Delft University of Technology, 1999.

[14] C. Décosse, W. A. Molnar and H. A. Proper. 'What Does DEMO Do? A Qualitative Analysis about DEMO in Practice: Founders, Modellers and Beneficiaries'. In: *Proceedings of the 4th Enterprise Engineering Working Conference (EEWC 2014), Funchal, Madeira*. Ed. by D. Aveiro, J. M. Tribolet and D. Gouveia. Vol. 174. Lecture Notes in Business Information Processing. Springer, Heidelberg, Germany, 2014, pp. 16–30. ISBN: 978-3-319-06504-5.

[15] D. Aveiro and D. D. Pinto. 'A Case Study Based New DEMO Way of Working and Collaborative Tooling'. In: *2013 IEEE 15th Conference on Business Informatics*. IEEE Computer Society Press, Los Alamitos, California, 2013, pp. 21–26. ISBN: 978-0-7695-5072-5. DOI: 10.1109/CBI.2013.12.

[16] S. de Kinderen, K. Gaaloul and H. A. Proper. 'On Transforming DEMO Models to ArchiMate'. In: *Enterprise, Business-Process and Information Systems Modeling - 13th International Conference, BPMDS 2012, 17th International Conference, EMMSAD 2012, and 5th EuroSymposium, held at CAiSE 2012, Gdańsk, Poland, June 25-26, 2012. Proceedings*. Ed. by I. Bider et al. Vol. 113. Lecture Notes in Business Information Processing. Gdańsk, Poland: Springer, Heidelberg, Germany, June 2012, pp. 270–284. ISBN: 978-3-642-31071-3.

[17] Artur Caetano, Aurélio Assis and José Tribolet. 'Using DEMO to analyse the consistency of business process models'. In: *Advances in Enterprise Information Systems II*. CRC Press, June 2012, pp. 133–146. DOI: 10.1201/b12295-17.

[18] Marné De Vries and Dominik Bork. 'Identifying Scenarios to Guide Transformations from DEMO to BPMN.' In: *EEWC*. 2020, pp. 92–110.

[19] Mark A T Mulder. 'NEN 7513 modelled in DEMO'. In: *Unpublished manuscript* (2021). URL: https://teec2.nl/nen7513/.

[20] 'ISO/IEC/IEEE 24765'. In: *Systems and Software Engineering – Vocabulary (SEVocab)* (2017).

[21] J. B. F. Mulder. *Rapid Enterprise Design*. 2007. ISBN: 9-081-04801-5.

[22] Domenik Bork and Hans-Georg Fill. 'Formal aspects of enterprise modeling methods: a comparison framework'. In: *2014 47th Hawaii international conference on system sciences*. IEEE. 2014, pp. 3400–3409.

[23] K. Peffers et al. 'A Design Science Research Methodology for Information Systems Research'. In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77.

[24] Roel J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[25] J. L. G. Dietz. *Enterprise Ontology – Theory and Methodology*. Springer, Heidelberg, Germany, 2006. ISBN: 978-3-540-29169-5.

[26] A. P. C. Perinforma. *The Essence of Organisation*. 3rd revised edition. The Netherlands: Sapio, 2013.

[27] J. L. G. Dietz. *DEMO Specification Language 3.6*. 2017.

[28] C. K. Ogden and I. A. Richards. *The Meaning of Meaning – A Study of the Influence of Language upon Thought and of the Science of Symbolism*. Oxford, United Kingdom: Magdalene College, University of Cambridge, 1923.

[29] Uwe Aßmann, Steffen Zschaler and Gerd Wagner. 'Ontologies, meta-models, and the model-driven paradigm'. In: Ontologies for software engineering and software technology. Springer, 2006, pp. 249–273.

[30] Lambertus Johannes Hommes. *The evaluation of business process modeling techniques*. TU Delft, Delft University of Technology, 2004.

[31] Todd Partridge. *What Exactly Does "Enterprise-Grade" Mean?* Website. July 2017. URL: https://www.linkedin.com/pulse/what-exactly-does-enterprise-grade-mean-todd-partridge.

[32] Gartner. *Information Technology Glossary*. website. Nov. 2020. URL: https://www.gartner.com/en/information-technology/glossary/enterprise-grade.

[33] Gerald Sparks. *What does Enterprise Grade mean?* website. Aug. 2020. URL: https://www.quora.com/What-does-Enterprise-Grade-mean.

[34] Ben Kepes. *What Does Enterprise GradeReally Mean?* website. Dec. 2013. URL: https://www.forbes.com/sites/benkepes/2013/12/18/what-does-enterprise-grade-really-mean.

[35] Raymond Nunn. *What is "Enterprise Grade Software"?* website. Nov. 2015. URL: http://tractsystems.com/what-is-enterprise-grade-software/.

[36] M. Op 't Land et al. *Enterprise Architecture - Creating Value by Informed Governance*. The Enterprise Engineering Series. Springer, Heidelberg, Germany, 2008. ISBN: 978-3-540-85231-5.

[37] R. Wagter and H. A. Proper. 'Involving the right stakeholders – Enterprise coherence governance'. In: *Architectural Coordination of Enterprise Transformation*. Ed. by H. A. Proper et al. The Enterprise Engineering Series. Springer, Heidelberg, Germany, 2018. Chap. 10, pp. 99–110. ISBN: 978-3-319-69583-9.

[38] Duarte Gouveia and David Aveiro. 'Things, References, Connectors, Types, Variables, Relations and Attributes–A Contribution to the FI and MU Theories'. In: Advances in Enterprise Engineering X. Springer, 2016, pp. 181–195.

[39] S. J. H. Van Kervel. 'Ontology driven Enterprise Information Systems Engineering'. en. ID: urn:NBN:nl:ui:24-uuid:8c42378a-8769-4a48-a7fb-f5457ede0759; ths:Dietz, J.L.G. - org:TU Delft - dgg:TU Delft, Delft University of Technology -. Doctoral dissertation. 2012. ISBN: 978-90-9027133-0. URL: http://resolver.tudelft.nl/uuid:8c42378a-8769-4a48-a7fb-f5457ede0759.

[40] Richard Feynman. 'EBNF: A Notation to Describe Syntax'. In: - (2016). URL: http://www.%20ics.%20uci.%20edu/~%20pattis/misc/ebnf2.%20pdf.

[41] D. L. Moody. 'The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering'. In: *IEEE Transactions on Software Engineering* 35.6 (2009), pp. 756–779.

[42] M.-E. Iacob et al. *ArchiMate 1.0 Specification*. The Open Group, 2009. ISBN: 978-9-087-53502-5.

[43] M. Op 't Land and J. L. G. Dietz. 'Enterprise ontology based splitting and contracting of organizations'. In: *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008), Fortaleza, Ceará, Brazil*. Ed. by L. M. Liebrock. ACM Press, New York, New York, 2008. ISBN: 978-1-59593-753-7.

[44] M. M. Lankhorst et al. 'Viewpoints and Visualisation'. In: pp. 171–214.

[45] Magno Andrade, David Aveiro and Duarte Pinto. 'Bridging Ontology and Implementation with a New DEMO Action Meta-model and Engine'. In: *Enterprise Engineering Working Conference*. Springer. 2019, pp. 66–82.

[46] James Cadle, Debra Paul and Paul Turner. *Business analysis techniques: 72 essential tools for success*. BCS, The Chartered Institute, 2010.

[47] Randy Goebel et al. 'Explainable AI: the new 42?' In: *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 2018, pp. 295–303.

[48] Thomas Allweyer. *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2016.

[49] Terry Halpin. 'ORM 2 graphical notation'. In: *Technical Report ORM2–02* (2005).

[50] Koen Smit, Martijn Zoet and Matthijs Berkhout. 'Functional Requirements for Business Rules Management Systems'. In: (2017).

[51] Roland Ettema and Jan LG Dietz. 'Archimate and demo–mates to date?' In: *Advances in Enterprise Engineering III*. Lecture Notes in Business Information Processing 34 (2009). Ed. by A. Albani, J. Barjis and J. L. G. Dietz, pp. 172–186.

[52] J. L. G. Dietz. *Enterprise Engineering The Manifesto*. Tech. rep. 2011. URL: `http://www.ciaonetwork.org/publications/EEManifesto.pdf`.

[53] J. L. G. Dietz. *Architecture: building strategy into design*. Academic Service The Hague, 2008.

[54] J.L.G. Dietz et al. 'The Discipline of Enterprise Engineering'. In: *International Journal of Organisational Design and Engineering* 3.1 (2013), pp. 86–114.

[55] Jan AP Hoogervorst and Jan LG Dietz. 'Enterprise architecture in enterprise engineering'. In: *Enterprise Modelling and Information Systems Architectures (EMISAJ)* 3.1 (2008), pp. 3–13.

[56] Donald H. Liles et al. 'Enterprise engineering: A discipline?' In: *Society for Enterprise Engineering Conference Proceedings*. Vol. 6. Citeseer, 1995. Chap. 1195.

[57] Paul Cilliers. 'Knowing complex systems'. In: *Managing Organisational Complexity: Philosophy, Theory, Application, IAP, Connecticut* (2005), pp. 7–19.

[58] Douglas O. Norman and Michael L. Kuras. 'Engineering complex systems'. In: Complex Engineered Systems. Springer, 2006, pp. 206–245.

[59] Peter Bernus et al. 'Enterprise engineering and management at the crossroads'. In: *Computers in Industry* 79 (2016), pp. 87–102.

[60] Howard Smith and Peter Fingar. *Business process management: the third wave*. Vol. 1. Meghan-Kiffer Press Tampa, FL, 2003.

[61] Traci A. Carte and Mark E. Cornelius. 'Integrating ERD and UML Concepts When Teaching Data Modeling.' In: *Journal of Information Systems Education* 17.1 (2006).

[62] Jürgen Habermas. *Theorie des kommunikativen Handelns*. Vol. 2. Suhrkamp Frankfurt, 1981.

[63] Ondřej Mráz et al. 'Converting DEMO PSI transaction pattern into BPMN: a complete method'. In: *Enterprise Engineering Working Conference*. Springer, 2017, pp. 85–98.

[64] Thomas Gray, Dominik Bork and Marné De Vries. 'A New DEMO Modelling Tool that Facilitates Model Transformations'. In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2020, pp. 359–374.

[65] Duarte Gouveia and David Aveiro. 'Modeling the system described by the EU General Data Protection Regulation with DEMO'. In: *Enterprise Engineering Working Conference*. Springer, 2018, pp. 144–158.

[66] J.L.G. Dietz and J.A.P. Hoogervorst. 'The principles of enterprise engineering'. In: *Enterprise Engineering Working Conference*. Springer, 2012, pp. 15–30.

[67] M. Dumay et al. 'Evaluation of DEMO and the Language/Action Perspective after 10 years of experience'. In: *Proceedings of LAP* (2005).

[68] V. E. Van Reijswoud. 'The structure of business communication: Theory, model and application.' Doctoral dissertation. 1996.

[69] J. L. G. Dietz. *DEMO-3 summary*. 2016.

[70] L. Terlouw. 'Modularization and Specification of Service-Oriented Systems'. Doctoral dissertation. Delft, the Netherlands: Delft Technical University, July 2011. ISBN: 978-9-461-08182-7.

[71] J. De Jong. 'A Method for Enterprise Ontology based Design of Enterprise Information Systems'. Doctoral dissertation. 2013.

[72] Roland Wilfred Ettema. *Using triangulation in lean six sigma to explain quality problems*. [Sl: sn], 2016.

[73] A. R. Hevner et al. 'Design science in information systems research'. In: *MIS quarterly* 28.1 (2004), pp. 75–105.

[74] J. van Aken and D. Andriessen. *Handboek ontwerpgericht wetenschappelijk onderzoek. [Handbook for Design Science Research]*. Boom Lemma, 2011.

[75] Stephan Kurpjuweit and Robert Winter. 'based meta model engineering'. In: *Enterprise modelling and information systems architectures–concepts and applications* (2007).

[76] Yan Wang. 'Transformation of demo models into exchangeable format'. In: *Delft University of Technology, Netherlands* (2009).

[77] Carlos Alberto da Silva Figueira. 'DEMO models based automatic workflow process generation'. In: (2013).

[78] E. Maij et al. 'Use cases and DEMO: aligning functional features of ICT-infrastructure to business processes'. In: *International journal of medical informatics* 65.3 (2002), pp. 179–191.

[79] Marien R. Krouwel and Martin Op't Land. 'Combining DEMO and Normalized Systems for developing agile enterprise information systems'. In: *Advances in Enterprise Engineering V* (2011), pp. 31–45.

[80] Linda Iris Terlouw. *Modularization and specification of service-oriented systems*. PhD thesis, Delft Technical University, Delft, The Netherlands, 2011.

[81] Arthur HM ter Hofstede and Henderik Alex Proper. 'How to formalize it?: Formalization principles for information system development methods'. In: *Information and Software technology* 40.10 (1998), pp. 519–540.

[82] J. L. G. Dietz. *DEMO Specification Language 3.5*. 2016.

[83] J. L. G. Dietz. *DEMO Specification Language 3.4*. 2016.

[84] Leo Apostel. 'Towards the formal study of models in the non-formal sciences'. In: *Synthese* (1960), pp. 125–161.

[85] Christos G. Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer Science and Business Media, 2009.

[86] Y. Wang, A. Albani and J. Barjis. 'Transformation of DEMO metamodel into XML schema'. In: Advances in Enterprise Engineering V. Springer, 2011, pp. 46–60.

[87] S. de Kinderen, K. Gaaloul and H. A. Proper. 'Transforming Transaction Models into ArchiMate'. In: EP-2012-Kirikova-CAiSEForum, pp. 114–121.

[88] S. de Kinderen, K. Gaaloul and H. A. Proper. 'Bridging value modelling to ArchiMate via transaction modelling'. In: *Software & Systems Modeling* 13.3 (2014), pp. 1043–1057.

[89] Y. Wang. 'Transformation of demo models into exchangeable format'. Doctoral dissertation. Delft University of TechnologyDelft, The Netherlands, 2009.

[90] Lucas O. Meertens et al. 'Mapping the business model canvas to archimate'. In: *Proceedings of the 27th annual ACM symposium on applied computing*. ACM, 2012, pp. 1694–1701.

[91] Antonio Gonçalves, Pedro Sousa and Marielba Zacarias. 'Capturing Activity Diagrams from Ontological Model'. In: *International Journal of Research in Business and Technology* 2.3 (2013), pp. 33–44.

[92] J. Vos. *Business modeling software focused on DEMO; http://wiki.xemod.eu*. 2011. URL: http://wiki.xemod.eu.

[93] Software AG. *ARIS http://www2.softwareag.com/*.

[94] CaseWise. *The Casewise Suite and Casewise Modeler; http://www.casewise.com/product/modeler/*. 2016. URL: http://www.casewise.com/product/modeler/.

[95] T. Severien. *Business Fundamentals - Verbeteren vanuit de essentie; http://www.businessfundamentals.nl/*. 2016. URL: http://www.businessfundamentals.nl/.

[96] Formetis. *Online modeling tool for process design and animation; https://www.demoworld.nl/*. 2017. URL: https://www.demoworld.nl/Portal/Home.

[97] Sparx. *Enterprise Architect https://www.sparxsystems.eu/start/home/*. 2017. URL: https://www.sparxsystems.eu/start/home/.

[98] Lambertus Johannes Hommes. *ModelWorld; http://ModelWorld.nl*. 2015.

[99] uSoft. *URequire Studio; http://www.usoft.com/software/urequire-studio*. 2016. URL: http://www.usoft.com/software/urequire-studio.

[100] R. Ryan Nelson. 'IT project management: Infamous failures, classic mistakes, and best practices.' In: *MIS Quarterly executive* 6.2 (2007).

[101] J. L. G. Dietz and J. Barjis. 'Supporting the DEMO Methodology with a Business Oriented Petri Net'. In: *Fourth CAiSE/ IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'99)*. Heidelberg, Germany, 1999.

[102] Abhimanyu Gupta, Geert Poels and Palash Bera. 'A Proposal of Using Conceptual Models for User Story Development and Maintenance'. In: *17th AIS SIGSAND Symposium*. 2018.

[103] Theo Janssen. *Enterprise Engineering: Sustained Improvement of Organizations*. Springer, 2015.

[104] M. Op 't Land and J.L.G. Dietz. 'Benefits of enterprise ontology in governing complex enterprise transformations'. In: *Enterprise Engineering Working Conference*. Springer, 2012, pp. 77–92.

[105] M. A. T. Mulder and H. A. Proper. 'Towards Enterprise-Grade Tool Support for DEMO'. In: *The Practice of Enterprise Modeling - 13th IFIP Working Conference, PoEM 2020, Riga, Latvia, November 25-27, 2020, Proceedings*. Ed. by Dominik Bork Jānis Grabis. Lecture Notes in Business Information Processin. Springer, 2020.

	80	Data		Exchange		Logic		Grammar		Verification		MDG		Shapescript	
		type	page	type	page	type	page	type	page	type	page	type	page	type	page
DEMO	Model	fig. 4.5	T-82	listing D.2	A-59	eq. E.1	A-106	-		-		fig. G.1	A-137	-	
	Schema	fig. 4.5	T-82	listing D.1	A-59	eq. E.1	A-106	-		-		fig. G.1	A-137	-	
	Aspectmodel	fig. 4.5	T-82	-		-		-		-		fig. G.2	A-137	-	
	CM	fig. 4.6	T-83	-		eq. E.2	A-107	-		appendix F.1	A-116	fig. G.5	A-138	-	
						eq. E.3	A-107								
						eq. E.4	A-107								
	PM	fig. 4.6	T-83	-		eq. E.35	A-111	-		-		fig. G.19	A-148	-	
						eq. E.36	A-111								
						eq. E.37	A-111								
	FM	fig. 4.6	T-83	-		eq. E.49	A-113	-		-		fig. G.20	A-148	-	
						eq. E.50	A-113								
						eq. E.51	A-113								
	AM	fig. 4.6	T-83			eq. E.63	A-115	listing C.6	A-17	appendix F.2	A-118	fig. G.21	A-149	-	
ETK	Guid	fig. 4.12	T-92	listing D.37	A-72	-		-		-		-		-	
	Id	fig. 4.12	T-92	listing D.55	A-76	-		listing C.1	A-14	-		fig. G.5	A-138	listing G.1	A-149
	Name	fig. 4.12	T-92	listing D.65	A-85	-		listing C.2	A-14	-		fig. G.5	A-138	listing G.1	A-149
	[e] executor	fig. 4.15	T-94	listing D.81	A-89	eq. E.7	A-107	-		-		fig. G.5	A-138	listing G.26	A-183
				listing D.82	A-89			-		-					
	ETK exists	fig. 4.15	T-94	-		eq. E.13	A-108	-		-		-		-	
	[i] initiator	fig. 4.15	T-94	listing D.79	A-88	eq. E.8	A-107	-		listing F.1	A-116	fig. G.5	A-138	listing G.25	A-182
				listing D.80	A-88			-		-					
	[c] TK in ATK	fig. 4.12	T-92	-		eq. E.9	A-107	-		-		-		-	
	[c] TK in CAR	fig. 4.12	T-92	-		eq. E.10	A-107	-		-		-		-	
				-		eq. E.11	A-108	-		-		-		-	
				-		eq. E.12	A-108	-		-		-		-	
ATK	[a] access to bank	fig. 4.14	T-94	listing D.28	A-70	eq. E.16	A-108	-		-		-		listing G.2	A-154
	FT in TK	fig. 4.14	T-94	-		eq. E.59	A-114	-		-		-		-	
EAR	EAR in CAR	fig. 4.15	T-94	listing D.29	A-70	eq. E.18	A-108	-		-		-		listing G.3	A-155
	exec single	fig. 4.15	T-94	-		eq. E.19	A-108	-		-		-		-	
	AR access ETK	fig. 4.15	T-94	-		eq. E.20	A-109	-		-		fig. G.5	A-138	-	
	exec multi	fig. 4.16	T-95	-		eq. E.21	A-109	-		-		-		-	
CAR		fig. 4.17	T-96	listing D.30	A-70	eq. E.22	A-109	-		-		-		listing G.4	A-160
		fig. 4.17	T-96	-		eq. E.23	A-109	-		-		-		-	
		fig. 4.17	T-96	-		eq. E.24	A-109	-		-		-		-	
		fig. 4.17	T-96	-		eq. E.25	A-109	-		-		-		-	
		fig. 4.17	T-96	-		eq. E.26	A-109	-		-		-		-	
ET		fig. 4.29	T-104	listing D.33	A-71	-		-		-		-		listing G.7	A-166
	[o] concerns	-		-		-		-		-		-		-	
	[s] specialisation	-		listing D.87	A-90	-		-		-		-		-	
	[r] aggregation	-		listing D.88	A-90	-		-		-		-		-	
	[g] generalisation	-		listing D.89	A-90	-		-		-		-		-	
PT		-		listing D.34	A-71	-		-		-		-		listing G.9	A-168
AT		-		listing D.35	A-71	-		-		-		-		-	
CET		-		-		-		-		-		-		-	
GSK	request	fig. 4.24	T-100	-		eq. E.44	A-111	-		-		-		listing G.5	A-161
	single	fig. 4.24	T-100	-		eq. E.46	A-112	-		-		-		-	
TPSK	TPSK part of TK	fig. 4.25	T-101	listing D.31	A-70	eq. E.47	A-112	-		-		-		listing G.6	A-161
	TPSK init TPSK	fig. 4.25	T-101	listing D.93	A-91	eq. E.48	A-112	-		-		-		listing G.29	A-186
	[y] wait	-		listing D.92	A-91	-		-		-		-		listing G.30	A-187
AR		-		-		-		listing C.4	A-15	-		-		-	
		-		-		-		listing C.5	A-15	-		-		-	
		-		-		-		-		-		-		-	
		-		-		-		-		-		-		-	
		-		-		-		-		-		-		-	
		-		-		-		-		-		-		-	

	80	Data		Exchange		Logic		Grammar		Verification		MDG		Shapescript	
		type	page	type	page	type	page	type	page	type	page	type	page	type	page
		-		-		-		-		-		-		-	
		-		-		-		-		-		-		-	
	[h] when	-		-		-		-		-		-		listing G.11	A-170
	[w] while	-		-		-		-		-		-		listing G.12	A-170
	[t] then	fig. 4.30	T-106	-		-		-		-		-		listing G.14	A-171
	[l] else	-		-		-		-		-		-		listing G.15	A-171
	[W] with	-		-		-		-		-		-		listing G.40	A-194
	[x] excludes	-		listing D.91	A-91	-		-		-		-		-	
		-		listing D.90	A-91	-		-		-		-		-	
Diagrams	OCD	fig. 4.11	T-90	listing D.108	A-94	eq. E.27	A-109	-		-		fig. G.18	A-148	-	
		fig. 4.11	T-90	-		eq. E.28	A-109	-		-		fig. G.12	A-144	-	
		fig. 4.11	T-90	-		eq. E.29	A-110	-		-		fig. G.12	A-144	-	
		fig. 4.11	T-90	-		eq. E.30	A-110	-		-		fig. G.12	A-144	-	
	TPT	-		-		eq. E.31	A-110	-		-		-		-	
		-		-		eq. E.32	A-110	-		-		-		-	
		-		-		eq. E.7	A-107	-		-		-		-	
		-		-		eq. E.56	A-114	-		-		-		-	
	BCT	-		-		eq. E.33	A-110	-		-		-		-	
		-		-		eq. E.34	A-110	-		-		-		-	
		-		-		eq. E.59	A-114	-		-		-		-	
	PSD	fig. 4.21	T-98	listing D.109	A-97	eq. E.38	A-111	-		-		fig. G.19	A-148	-	
		fig. 4.21	T-98	-		eq. E.39	A-111	-		-		fig. G.14	A-146	-	
		fig. 4.21	T-98	-		eq. E.40	A-111	-		-		-		-	
	TPD	fig. 4.23	T-99	listing D.110	A-98	eq. E.41	A-111	-		-		fig. G.19	A-148	-	
		fig. 4.23	T-99	-		eq. E.42	A-111	-		-		fig. G.16	A-147	-	
	OFD	-		listing D.111	A-98	eq. E.54	A-113	-		-		fig. G.20	A-148	-	
		-		-		eq. E.53	A-113	-		-		fig. G.14	A-146	-	
	ARD	fig. 4.30	T-106	listing D.112	A-99	eq. E.66	A-115	-		-		fig. G.21	A-149	-	
		fig. 4.30	T-106	-		eq. E.68	A-115	-		-		fig. G.11	A-143	-	
		fig. 4.30	T-106	-		eq. E.69	A-115	-		-		fig. G.11	A-143	-	
		-		-		-		-		-		-		-	
	WIS	-		-		-		-		-		fig. G.21	A-149	listing G.16	A-172
		-		-		-		-		-		fig. G.17	A-147	listing G.42	A-196
	ARS	-		listing D.112	A-99	-		-		-		fig. G.21	A-149	-	
		-		-		-		-		-		fig. G.11	A-143	-	
	AFD	-		listing D.113	A-100	-		-		-		fig. G.10	A-142	-	
		-		-		-		-		-		fig. G.18	A-148	-	
	DFS	-		-		-		-		-		fig. G.20	A-148	-	
	ELS	-		-		-		-		-		fig. G.20	A-148	-	
		-		-		-		-		-		-		-	
	SEC	-		-		-		-		-		fig. G.15	A-146	-	

List of Figures

List of Tables

List of Equations

Listings

Part IV

Appendix

Appendix A

Validation Example

The model in this chapter is used as a validation model. This model has all aspects of the DEMOSL proposal and has all faults that can be detected. Due to the complexity of the specification language some errors are duplicated in the model to force other errors.

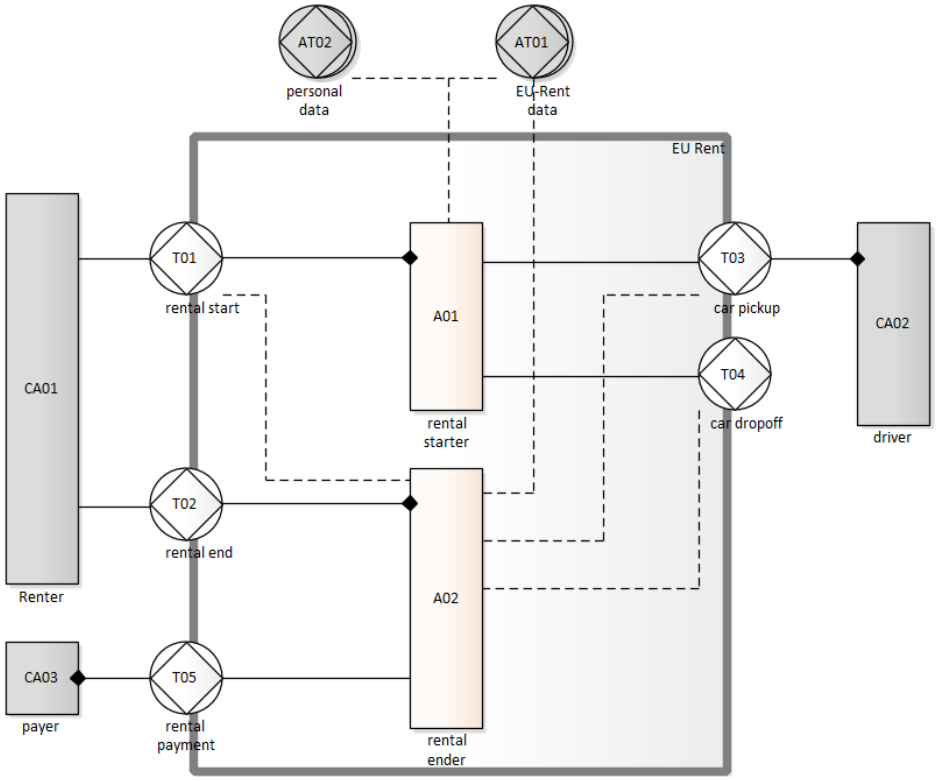


Figure A.1: EU Rental

The Organisation Construction Diagram (OCD) of the often used EU Rental case can already be modelled using Sparx Enterprise Architect (SEA). We used the tool to model EU Rental in fig. A.1 as modelled in the example of ModelWorld [1].
The visualisation and the model of the OCD comply to the DEMO Specification Language (DEMOSL) specifications. The connections and elements are drawn within specifications

[1] [98] - Lambertus Johannes Hommes. *ModelWorld; http://ModelWorld.nl.* 2015

of the visualisation. The validation has been run on this model and has returned no errors in this model.

Appendix B

DEMOSL 3.7 Grammar

```
// DEMOSL 3.7 Slide 5 line 20-23
// RESERVED WORDS
// Reserved words are terms that have a definite meaning in DEMO. They are
    always underlined. Next to the 'present␣tense␣intention␣terms' and the '
    perfect␣tense␣intention␣terms' (specified above), the next reserved words
    exist: new, performer, addressee, et (event time), st (settlement time)

request promise state accept decline quit reject stop revoke allow refuse
requested promised stated accepted declined quitted rejected stopped revoked
    allowed refused
new, performer, addressee, et, st
```

Listing B.1: DEMOSL Reserved words

```
// DEMOSL 3.7 Slide 4 line 1
// transaction kind id= "T", {digit}-; //Example: T17
transaction_kind_id: TransactionPrefix Number ;

// DEMOSL 3.7 Slide 4 line 2
// aggregate transaction kind id= "AT", {digit}-; //Example: AT2
aggregate_transaction_kind_id: AggregateTransactionPrefix Number;

// DEMOSL 3.7 Slide 4 line 3
// actor role id= "A", {digit}-; //Example: A17
actor_role_id: ActorPrefix Number;

// DEMOSL 3.7 Slide 4 line 4
// composite actor role id= "CA", {digit}-; //Example: CA1
composite_actor_role_id: CompositeActorPrefix Number;

// DEMOSL 3.7 Slide 4 line 5
// fact kind id= "F", {digit}-; //Example: F17
fact_kind_id: FactPrefix Number;
```

```
// DEMOSL 3.7 Slide 4 line 6
// product kind id= "P", {digit}-; //Example: P17
product_kind_id: ProductPrefix Number;
```

Listing B.2: Action Rule element Ids

```
// DEMOSL 3.7 Slide 4 line 7
// transaction kind name = {lower case letter}-, {"␣", {lower case letter}}; //
   Example: rental concluding
transaction_kind_name : Lower_case_word (Lower_case_word)*;

// DEMOSL 3.7 Slide 4 line 8
// actor role name = {lower case letter}-, {"␣", {lower case letter}}; //
   Example: rental concluder
actor_role_name : Lower_case_word (Lower_case_word)*;

// DEMOSL 3.7 Slide 4 line 9
// object class name = {upper case letter}-, {"␣", {upper case letter}}; //
   Examples: RENTAL, CAR GROUP
object_class_name : Upper_case_word (Upper_case_word)*;

// DEMOSL 3.7 Slide 4 line 10
// fact kind name = {lower case letter}-, {"␣", {lower case letter}}; //
   Examples: member, daily rental rate
fact_kind_name : Lower_case_word (Lower_case_word)*;
```

Listing B.3: Action Rule element names

```
// DEMOSL 3.7 Slide 5 line 12-15
// product kind formulation= entity variable | property variable, "is", perfect
    tense sentence;
// Example: Rental is contracted
// Example: the first fee of Membership is paid
// Example: the fee of Membership in Year is paid
product_kind_formulation:
       (entity_variable
       | property_variable)
       'is'
       perfect_tense_sentence
       ;
```

Listing B.4: Action Rule product kind formulation

```
// DEMOSL 3.7 Slide 4 line 11
entity_variable : definite_entity_variable | indefinite_entity_variable |
   property_variable;

// DEMOSL 3.7 Slide 4 line 12-14
```

```
// definite entity variable = upper case letter, {lower case letter}, {"␣",
    upper case letter, {lower case letter}} | "[", {lower case letter}-, {"␣",
    {lower case letter}}, "]"
// Examples: Person, [person], Car Group, [car group], Year, [year]
definite_entity_variable:
       Capital_case_word (Capital_case_word)*
       | BracketPre Lower_case_word (Lower_case_word)* BracketPost;

// DEMOSL 3.7 Slide 4 line 15-16
// indefinite entity variable = "some", object class name; //Examples: some
    PERSON, some YEAR
indefinite_entity_variable : SomeKeyWord object_class_name;

// DEMOSL 3.7 Slide 4 line 17-20
// property variable = "the", property kind name, "of" | "in" | "on", definite
    entity variable;
// Example: the member of Membership, the member of [membership]
// Example: the daily rental rate of Car Group in Year, the daily rental rate
    of [car group] in [year]
// Example: the number of cars in Car Group on Day, the number of cars in [car
    group] on [day]
property_variable:
       'the' property_kind_name
       ('of' | 'in' | 'on')
       definite_entity_variable;

// DEMOSL 3.7 Slide 4 line 21-22
// product variable = < definite entity variable regarding a product >;
// Examples: Membership, [membership], Rental, [rental]
product_variable : '<definite␣entity␣variable␣regarding␣a␣product>';
```

Listing B.5: Action Rule Entity Variables

```
// DEMOSL 3.7 Slide 14 line 1
// action rule specification = event part, assess part, response part;
action_rule_specification :
    event_part
    assess_part
    response_part
    ;
```

Listing B.6: Action Rule specification

```
// DEMOSL 3.7 Slide 14 line 2
// event part= agendum clause, [while clause], [with clause];
event_part:
    agendum_clause
    (while_clause)?
```

```
    (with_clause)?
    ;
```

Listing B.7: Action Rule event part

```
// DEMOSL 3.7 Slide 14 line 3
// agendum clause = "when", new transaction reference, "is", perfect tense
    intention;
agendum_clause :
    'when'
    new_transaction_reference
    'is'
    perfect_tense_intention
    ;
```

Listing B.8: Action Rule agendum

```
// DEMOSL 3.7 Slide 14 line 4
// while clause = "while", {transaction reference, "is", perfect tense
    intention, {with clause};}-;
while_clause :
    'while'
    ( transaction_reference
      'is'
      perfect_tense_intention
      (with_clause)
    )+;

// DEMOSL 3.7 Slide 14 line 13
// transaction reference = transaction kind name, "for", product object
    reference;
transaction_reference:
    transaction_kind_name
        'for'
        product_object_reference
        ;

// DEMOSL 3.7 Slide 14 line 14
// new transaction reference = transaction kind name, "for", ["new"], product
    object reference;
new_transaction_reference:
    transaction_kind_name
        'for'
        ('new')?
        product_object_reference;
```

Listing B.9: Action Rule while clause

```
// DEMOSL 3.7 Slide 14 line 5
// with clause = "with", {property kind formulation}-;
with_clause :
    'with'
    (property_kind_formulation)+;

// DEMOSL 3.7 Slide 5 line 2
// property kind formulation = property variable, "is", object variable;
property_kind_formulation :
    property_variable
    'is'
    object_variable
    ;
// DEMOSL 3.7 Slide 5 line 3-5
// Example: the member of Membership is Person
// Example: the daily rental rate of Car Group in Year is Money
// Example: the son of Person is some PERSON
```

Listing B.10: Action Rule with clause

```
// DEMOSL 3.7 Slide 14 line 6
// assess part = justice sub part, sincerity sub part, truth sub part;
assess_part:
    justice_sub_part
        sincerity_sub_part
        truth_sub_part
        ;

// DEMOSL 3.7 Slide 14 line 7
// justice sub part = "justice:", "<no␣specific␣condition>" | {fact kind
    formulation}-;
justice_sub_part:
    'justice:'
        ( '<no␣specific␣condition>'
          | (fact_kind_formulation)+
        )
        ;

// DEMOSL 3.7 Slide 14 line 8
// sincerity sub part = "sincerity:", "<no␣specific␣condition>" | {fact kind
    formulation}-;
sincerity_sub_part :
    'sincerity:'
        ( '<no␣specific␣condition>'
          | (fact_kind_formulation)+
        )
        ;
```

```
// DEMOSL 3.7 Slide 14 line 9
// truth sub part = "truth:", "<no␣specific␣condition>" | {fact kind
   formulation}-;
truth_sub_part :
   'truth:'
       ( NoSpecificCondition
         | (fact_kind_formulation)+
       )
       ;
```

Listing B.11: Action Rule Assess part

```
// DEMOSL 3.7 Slide 5 line 1
// fact kind formulation = property kind formulation | attribute kind
   formulation | product kind formulation;
fact_kind_formulation :
         property_kind_formulation
       | attribute_kind_formulation
       | product_kind_formulation
       ;
```

Listing B.12: Action Rule fact kind formulation

```
// DEMOSL 3.7 Slide 5 line 6
// attribute kind formulation = attribute variable, "is␣equal␣to" | "is␣greater
   ␣than" | "is␣less␣than", value variable;
attribute_kind_formulation :
   attribute_variable
       (
           'is␣equal␣to'
         | 'is␣greater␣than'
         | 'is␣less␣than'
       )
       value_variable
       ;

// DEMOSL 3.7 Slide 5 line 7
// attribute variable = < property variable regarding an abstract object (=
   value) >;
attribute_variable :
       '<property␣variable␣regarding␣an␣abstract␣object␣(:␣value)>'
       ;

// DEMOSL 3.7 Slide 5 line 8
// value variable = < object variable regarding an abstract object (= value) >
   | value;
value_variable:
```

```
          '<object␣variable␣regarding␣an␣abstract␣object␣(:␣value)>'
        | value
        ;

// DEMOSL 3.7 Slide 5 line 9
// value = [dimension, ":"], (real | integer, unit) | boolean; < Cf. slide 15 >
value :
    ( dimension ':' )?
        ( ( (
            Real
          | Integer
        ) Unit
          )
          | Boolean
        )
        ;
// DEMOSL 3.7 Slide 5 line 10-11
// Example: LENGTH: 2.4 m
// Example: 2.4 m
```

Listing B.13: Action Rule attribute kind formulation

```
// DEMOSL 3.7 Slide 14 line 10-11
// response part = "if", "complying␣with", present tense intention, "is␣
    considered␣justifiable", "then", action clause, ["else" action clause];
response_part :
        'if' 'complying␣with'
        present_tense_intention
        'is␣considered␣justifiable'
        'then' action_clause
        ('else' action_clause)?
        ;

// DEMOSL 3.7 Slide 14 line 12
// action clause = {present tense intention, transaction reference, {with
    clause}};
action_clause :
        present_tense_intention
        transaction_reference
        (with_clause)?
        ;
```

Listing B.14: Action Rule response part

```
// DEMOSL 3.7 Slide 5 line 16-17
// present tense intention = "request" | "promise" | "state" | "accept" | "
    decline" | "quit" | "reject" | "stop" | "revoke" | "allow" | "refuse";
```

```
present_tense_intention : 'request' | 'promise' | 'state' | 'accept' | 'decline
    ' | 'quit' | 'reject' | 'stop' | 'revoke' | 'allow' | 'refuse';

// DEMOSL 3.7 Slide 5 line 18-19
// perfect tense intention = "requested" | "promised" | "stated" | "accepted" |
     "declined" | "quitted" | "rejected" | "stopped" | "revoked" | "allowed" |
    "refused";
perfect_tense_intention : 'requested' | 'promised' | 'stated' | 'accepted' | '
    declined' | 'quitted' | 'rejected' | 'stopped' | 'revoked' | 'allowed' | '
    refused';
```

Listing B.15: Action Rule intentions

Appendix C

Grammar Metamodel

C.1 Element identification

```
transaction_kind_id:
    TransactionKindId
    ;

aggregate_transaction_kind_id:
    AggregateTransactionKindId
    ;

actor_role_id:
    ActorKindId
    ;

composite_actor_role_id:
    CompositeActorKindId
    ;

fact_kind_id:
    FactKindId
    ;

product_kind_id:
    ProductKindId
    ;
```

Listing C.1: Action Rule element identification

C.2 Element Names

```
transaction_kind_name : Lower_case_word (Lower_case_word)*;


actor_role_name : Lower_case_word (Lower_case_word)*;


object_class_name : Upper_case_word (Upper_case_word)*;


fact_kind_name : Lower_case_word (Lower_case_word)*;
```

Listing C.2: Action Rule Element names

C.3 Reserved Words

```
/* Cautious words */
/* for new of in on the */
```

Listing C.3: Grammar Reserved words

Within a grammar the whitespace and readability extra lines need to be skipped for finding the information in the rule text.

```
COMMENT : '
WHITESPACE:␣[␣\r\t\u000C\n]+␣->␣skip␣;
```

Listing C.4: Action Rule common rules

These types are needed to recognise the keywords in the grammar.

```
SomeKeyword: 'some' ;
RuleKeyword: 'rule' ;
WhenKeyword: 'when' ;
WhileKeyword: 'while' ;
WithKeyword: 'with' ;
IsKeyword: 'is' ;
ForKeyword: 'for' ;
EachKeyword: 'each' ;
NewKeyword: 'new' ;
OfKeyword: 'of' ;
InKeyword: 'in' ;
OnKeyword: 'on' ;
TheKeyword: 'the' ;
IfKeyword: 'if' ;
```

```
ElseKeyword: 'else' ;
ThenKeyword: 'then' ;
SetKeyword: 'set' ;

RequestKeyword: 'request' ;
PromiseKeyword: 'promise' ;
StateKeyword: 'state' ;
AcceptKeyword: 'accept' ;
DeclineKeyword: 'decline' ;
QuitKeyword: 'quit'  ;
RejectKeyword: 'reject' ;
StopKeyword: 'stop'  ;
RevokeKeyword: 'revoke' ;
AllowKeyword: 'allow' ;
RefuseKeyword: 'refuse' ;
ExecuteKeyword: 'execute' ;

RequestedKeyword: 'requested' ;
PromisedKeyword: 'promised' ;
StatedKeyword: 'stated' ;
AcceptedKeyword: 'accepted' ;
DeclinedKeyword: 'declined' ;
QuittedKeyword: 'quitted' ;
RejectedKeyword: 'rejected' ;
StoppedKeyword: 'stopped' ;
RevokedKeyword: 'revoked' ;
AllowedKeyword: 'allowed' ;
RefusedKeyword: 'refused' ;
ExecutedKeyword: 'executed' ;

RequestAbKeyword: 'rq' ;
PromiseAbKeyword: 'pm' ;
StateAbKeyword: 'st' ;
AcceptAbKeyword: 'ac' ;
DeclineAbKeyword: 'dc' ;
QuitAbKeyword: 'qt'  ;
RejectAbKeyword: 'rj' ;
StopAbKeyword: 'sp'  ;
RevokeAbKeyword: 'rv' ;
AllowAbKeyword: 'al' ;
RefuseAbKeyword: 'rf' ;
ExecuteAbKeyword: 'ex' ;

TrueKeyword: 'true' ;
FalseKeyword: 'false' ;
```

```
AccessKeyword: 'assess' ;
JusticeKeyword: 'justice' ;
SincirityKeyword: 'sincerity' ;
TruthKeyword: 'truth' ;

CompareEqualKeyword: 'equal' ;
CompareGreaterKeyword: 'greater' ;
CompareLessKeyword: 'less' ;
CompareThanKeyword: 'than' ;
CompareToKeyword: 'to' ;
CompareOrKeyword: 'or' ;
CompareAndKeyword: 'and' ;
TransactionKindId: 'T'([0-9]+'-')?[0-9]+ ;
CompositeActorKindId: 'CA'([0-9]+'-')?[0-9]+ ;
AggregateTransactionKindId: 'AT'([0-9]+'-')?[0-9]+ ;
FactKindId: 'F'([0-9]+'-')?[0-9]+ ;
ProductKindId: 'P'([0-9]+'-')?[0-9]+ ;
ActorKindId: 'A'([0-9]+'-')?[0-9]+ ;
DoubleQuote: '"' ;
BracketPre: '[' ;
BracketPost: ']' ;
LineEndKeyword: ';' ;
fragment Digit : [0-9] ;
fragment Digits : Digit+ ;
Integer: ('+'|'-')? Digits;
Real: ('+'|'-')? Digits '.' Digits;
Name: [a-zA-Z_][a-zA-Z0-9_]* ;
Lower_case_word: [a-z]+ ;
Upper_case_word: [A-Z]+;
```

Listing C.5: Action Rule base types

C.4 Grammar core

```
action_rule_specification :
    RuleKeyword
    bracketed_name
    event_part
    assess_part
    response_part
    ;
```

Listing C.6: Action Rule specification

```
event_part:
    agendum_clause
```

```
    (while_clause)?
    ;
```

Listing C.7: Action Rule event part

```
assess_part :
    (
        AccessKeyword
        justice_sub_part
        sincerity_sub_part
        truth_sub_part
        )?
        ;

justice_sub_part :
    (
        JusticeKeyword ':'
        (fact_kind_formulation)+
        )?
        ;

sincerity_sub_part :
    (
        SincirityKeyword ':'
        (fact_kind_formulation)+
        )?
        ;

truth_sub_part :
    (
        TruthKeyword ':'
        (
          foreach_clause
          '
␣␣␣␣␣␣␣␣␣␣fact_kind_formulation+
␣␣␣␣␣␣␣␣␣␣'}'␣\r\n");}
␣␣␣␣␣␣␣␣|␣fact_kind_formulation+
␣␣␣␣␣␣␣␣)
␣␣␣␣␣␣␣␣)?
␣␣␣␣␣␣␣␣;
```

Listing C.8: Action Rule assess part

```
response_part :
    IfKeyword 'complying␣with'
    present_tense_intention
    'is␣considered␣justifiable'
    ThenKeyword
```

```
       (
         foreach_clause ''
    | action_clause+
       )
    (ElseKeyword
    action_clause+)?
    ;

action_clause :
    present_tense_intention
    transaction_reference
    transaction_step_reference
    (with_clause)*
    ;
```

Listing C.9: Action Rule response part

```
agendum_clause :
    WhenKeyword
    transaction_reference
    IsKeyword
    perfect_tense_intention
    transaction_step_reference
    (with_clause)*
    ;
```

Listing C.10: Action Rule agendum part

```
with_clause:
    WithKeyword
    (property_kind_formulation)+
    ;


property_kind_formulation :
    property_variable
    IsKeyword
    (
      (SomeKeyword )? bracketed_name
    | property_variable
        | SetKeyword OfKeyword
          bracketed_name
          '
␣␣␣␣␣␣␣␣␣␣(with_clause)+
␣␣␣␣␣␣␣␣␣␣'}'␣␣");}
␣␣␣␣)␣';'
␣␣␣␣;
```

Listing C.11: Action Rule with clause

```
foreach_clause:
   ForKeyword EachKeyword
   bracketed_name
   InKeyword
   '
␣␣␣property_variable
␣␣␣'}'␣␣");}
␣␣␣;
```

Listing C.12: Action Rule for each clause

```
while_clause:
    WhileKeyword
        (
          foreach_clause
          '
␣␣␣␣␣␣␣␣␣␣(while_subclause)+
␣␣␣␣␣␣␣␣␣␣'}'␣␣");}
␣␣␣␣|␣while_subclause+
␣␣␣␣␣␣␣␣)
␣␣␣␣;

while_subclause:
␣␣␣␣transaction_reference␣IsKeyword␣␣perfect_tense_intention
␣␣␣␣transaction_step_reference
␣␣␣␣(with_clause)*
␣␣␣␣␣␣␣␣;

transaction_step_reference:
␣␣␣␣'('
␣␣␣␣transaction_kind_id
␣␣␣␣␣␣␣␣'/'
␣␣␣␣␣␣␣␣intention_abbreviation
␣␣␣␣␣␣␣␣')'
␣␣␣␣;
transaction_reference␣:




␣␣␣␣bracketed_name
␣␣␣␣ForKeyword
␣␣␣␣(NewKeyword)?
␣␣␣␣bracketed_name
␣␣␣␣;
```

Listing C.13: Action Rule while clause

```
property_variable :
    TheKeyword
    bracketed_name
    ('of' | 'in' | 'on')
    bracketed_name
    (( 'in' )
    bracketed_name
    )?
    ;
```

Listing C.14: Action Rule property variable

```
fact_kind_formulation :
      property_kind_formulation
    | attribute_kind_formulation

    ;
```

Listing C.15: Action Rule fact kind

```
attribute_kind_formulation :
    property_variable
    IsKeyword
    comparison
    value_variable
    ';'
    ;

comparison:
      operator CompareOrKeyword comparison
    | operator CompareAndKeyword comparison
    | operator
    ;

operator:
      CompareEqualKeyword CompareToKeyword
    | CompareGreaterKeyword CompareThanKeyword
    | CompareLessKeyword CompareThanKeyword
    | InKeyword
    ;

value_variable :
          property_variable
        | value
        ;

value :
```

```
       ( dimension ':' )?
       ( ( Real
          | Integer
       ) unit) | boolean
       ;

dimension:
       name
       ;

boolean:
     TrueKeyword
       | FalseKeyword
       ;

unit:
       name
       ;
```

Listing C.16: Action Rule attribute kind

```
present_tense_intention :
     'request'
   | 'promise'
   | 'state'
   | 'accept'
   | 'decline'
   | 'quit'
   | 'reject'
   | 'stop'
   | 'revokerequest'
      | 'revokerequestallow'
      | 'revokerequestrefuse'
   | 'revokepromise'
      | 'revokepromiseallow'
      | 'revokepromiserefuse'
   | 'revokedeclare'
      | 'revokedeclareallow'
      | 'revokedeclarerefuse'
   | 'revokeaccept'
      | 'revokeacceptallow'
      | 'revokeacceptrefuse'
   | 'execute'
      ;

intention_abbreviation :
     'rq'
```

```
    | 'pm'
    | 'st'
    | 'ac'
    | 'dc'
    | 'qt'
    | 'rj'
    | 'sp'
       | 'rvrq'
    | 'rvra'
    | 'rvrr'
    | 'rvpq'
    | 'rvpa'
    | 'rvpr'
    | 'rvdq'
    | 'rvda'
    | 'rvdr'
    | 'rvaq'
    | 'rvaa'
    | 'rvar'
    | 'ex'
       ;

perfect_tense_intention :
      'requested'
    | 'promised'
    | 'stated'
    | 'accepted'
    | 'declined'
    | 'quitted'
    | 'rejected'
    | 'stopped'
       | 'revokedrequest'
       | 'revokerequestallowed'
       | 'revokerequestrefused'
       | 'revokedpromise'
       | 'revokepromiseallowed'
       | 'revokepromiserefused'
       | 'revokeddeclare'
       | 'revokedeclareallowed'
       | 'revokedeclarerefused'
       | 'revokedaccept'
       | 'revokeacceptallowed'
       | 'revokeacceptrefused'
    | 'executed'
       ;
```

Listing C.17: Action Rule intentions

```
stringed_name: DoubleQuote name DoubleQuote ;

bracketed_name:
    BracketPre name
    BracketPost
    | name
    ;

name:
      (
        SomeKeyword
      | IsKeyword
      | SomeKeyword
      | RuleKeyword
      | WhenKeyword
      | WhileKeyword
      | WithKeyword
      | IsKeyword
      | ForKeyword
      | EachKeyword
      | NewKeyword
      | OfKeyword
      | InKeyword
      | OnKeyword
      | IfKeyword
      | ThenKeyword
      | ElseKeyword
      | SetKeyword

      | TransactionKindId
      | CompositeActorKindId
      | AggregateTransactionKindId
      | FactKindId
      | ProductKindId
      | ActorKindId

      | RequestKeyword
      | PromiseKeyword
      | StateKeyword
      | AcceptKeyword
      | DeclineKeyword
      | QuitKeyword
      | RejectKeyword
      | StopKeyword
      | RevokeKeyword
```

```
  | AllowKeyword
  | RefuseKeyword
  | ExecuteKeyword

  | RequestedKeyword
  | PromisedKeyword
  | StatedKeyword
  | AcceptedKeyword
  | DeclinedKeyword
  | QuittedKeyword
  | RejectedKeyword
  | StoppedKeyword
  | RevokedKeyword
  | AllowedKeyword
  | RefusedKeyword
  | ExecutedKeyword

  | RequestAbKeyword
  | PromiseAbKeyword
  | StateAbKeyword
  | AcceptAbKeyword
  | DeclineAbKeyword
  | QuitAbKeyword
  | RejectAbKeyword
  | StopAbKeyword
  | RevokeAbKeyword
  | AllowAbKeyword
  | RefuseAbKeyword
  | ExecuteAbKeyword

  | TheKeyword
  | AccessKeyword
  | JusticeKeyword
  | SincirityKeyword
  | TruthKeyword
  | CompareEqualKeyword
  | CompareGreaterKeyword
  | CompareLessKeyword
  | CompareThanKeyword
  | CompareToKeyword
  | CompareOrKeyword
  | CompareAndKeyword
  | TrueKeyword
  | FalseKeyword
  | Name
  ) (name )*
;
```

Listing C.18: Action Rule variable names

C.5 Action Rule References

All action rules that have been used in specifications and course material, that has been found, have been investigated and tested in the new rule specifications. All rules that needed adaptation have green additions and red deletions.

C.5.1 DB RAC A1/T1 new rq

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 41-42"

```
when rental concluding for new Rental
     is requested (T1/rq)
  with
    the starting day ... some day
    the ending day ... some day
    the renter ... some person
    the payer ... some person
    the driver ... some person
    the payer ... some person
    the car ... car group
    the issuing location ... some
        branch
    the return location ... some
        branch

assess
  justice:
    the performer ... of Rental;
    the addressee ... rental
        concluder;
  sincerity: <no specific condition>
  truth:
    the starting day of Rental is in
        the Rental Horizon of the year
         of the starting day of Rental
        ;
    the ending day ... of Rental;
    the ending day ... of Rental;
    the duration ... day of Rental;
    the number of cars in the car
        group of Rental on every day
        in the rental period of Rental
         is greater than zero
```

Listing C.19: DB RAC A1/T1 new rq Original

```
rule DB RAC A1_T1 new rq
when rental concluding for new Rental
     is requested (T1/rq)
  with
    the starting day ... some day;
    the ending day ... some day;
    the renter ... some person;
    the payer ... some person;
    the driver ... some person;
    the payer ... some person;
    the car ... car group;
    the issuing location ... some
        branch;
    the return location ... some
        branch;

assess
  justice:
    the performer ... of Rental;
    the addressee ... of rental;

  truth:
    the starting day of Rental is in
        the [Rental Horizon of the
        year of the starting day] of
        Rental;
    the ending day ... [Rental ...
        day] of Rental;
    the ending day ... of Rental;
    the duration ... the [max ... day
        ] of Rental;
    the [number of cars in the car
        group of Rental on every day
        in the rental period] of
        Rental is greater than 0
        cars;
```

Listing C.20: DB RAC A1/T1 new rq Adapted

Table C.1: Action Rule DB RAC A1/T1 new rq (event and access)

```
if complying with the assessment is
    considered justifiable
  then
  request rental paying for Rental
      [T2/rq]

    with
      the addressee of the request is
          the payer of Rental;
      the requested production day of
          rental paying for Rental is
          less than or equal to the
          starting day of Rental;

      the requested paid rental
          amount of Rental is equal
          to the rental charge of
          Rental;
    else
      decline rental concluding for
          Rental [T1/dc]
        with the addressee of the
            decline is the renter of
            Rental
```

Listing C.21: DB RAC A1/T1 new rq Original

```
if complying with request is
    considered justifiable
  then
    request rental paying for Rental
        (T2/rq)
    with
      the addressee of the request is
          the payer of Rental;
      the [requested production day
          of rental paying] of Rental
          is the starting day of
          Rental;


      the requested paid rental
          amount of Rental is the
          rental charge of Rental;
  else
    decline rental concluding for
        Rental (T1/dc)
    with
      the addressee of the decline is
          the renter of Rental;
```

Listing C.22: DB RAC A1/T1 new rq Adapted

Table C.2: Action Rule DB RAC A1/T1 new rq (result)

C.5.2 DB RAC A1/T1 rq

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 43"

Listing C.23: DB RAC A1/T1 rq Original

Listing C.24: DB RAC A1/T1 rq Adapted

Table C.3: Action Rule DB RAC A1/T1 rq

C.5.3 DB RAC A1/T2 st

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 44"

```
when rental paying for Rental is
    stated (T2/st)

assess
  justice:
    the performer of the statement is
        the renter of Rental;
    the addressee of the statement is
        the rental concluder of
       Rental;
   sincerity: <no specific condition>
   truth:
     the stated paid rental amount of
         Rental is equal to the
        promised paid rental amount
        of Rental;
     the stated production day of
        rental paying for Rental is
        equal to the promised
        production day of rental
        paying for Rental

if complying with the assessment is
    considered justifiable
then
  accept rental paying for Rental [T2
      /ac]
  with
    the addressee of the acceptance
       is the renter of Rental
else
  reject rental paying for Rental [T2
      /rj]
  with
    the addressee of the rejection is
        the renter of Rental
```

Listing C.25: DB RAC A1/T2 st Original

```
rule DB RAC A1_T2 st
when rental paying for Rental is
    stated (T2/st)

assess
  justice:
    the performer of the statement is
        the renter of Rental;
    the addressee of the statement is
        the rental concluder of
       Rental;

   truth:
     the stated paid rental amount of
         Rental is equal to the
        promised paid rental amount
        of Rental;
     the stated production day of
        rental paying for Rental is
        equal to the promised
        production day of rental
        paying for Rental;

if complying with state is considered
     justifiable
then
  accept rental paying for Rental (T2
      /ac)
  with
    the addressee of the acceptance
       is the renter of Rental;
else
  reject rental paying for Rental (T2
      /rj)
  with
    the addressee of the rejection is
        the renter of Rental;
```

Listing C.26: DB RAC A1/T2 st Adapted

Table C.4: Action Rule DB RAC A1/T2 st

C.5.4 DB RAC A1/T1 pm

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 45"

```
when rental concluding for Rental is
    promised (T1/pm)
  while rental paying for Rental is
      accepted (T2/ac)
assess
  justice:
    the performer of the promise is
        the rental concluder of Rental
        ;
    the addressee of the promise is
        the renter of Rental;
  sincerity: <no specific condition>
  truth:  <no specific condition>
if complying with the assessment is
    considered justifiable
then
  execute rental concluding for
      Rental [T1/ex]
  state rental concluding for Rental
      [T1/st]
  with
    the addressee of the statement is
        the renter of Rental
```

Listing C.27: DB RAC A1/T1 pm Original

```
rule DB RAC A1_T1 pm
when rental concluding for Rental is
    promised (T1/pm)
  while rental paying for Rental is
      accepted (T2/ac)
assess
  justice:
    the performer of the promise is
        the rental concluder of Rental
        ;
    the addressee of the promise is
        the renter of Rental;


if complying with promise is
    considered justifiable
then
  execute rental concluding for
      Rental (T1/ex)
  state rental concluding for Rental
      (T1/st)
  with
    the addressee of the statement is
        the renter of Rental;
```

Listing C.28: DB RAC A1/T1 pm Adapted

Table C.5: Action Rule DB RAC A1/T1 pm

C.5.5 DB RAC A3/T3 rq

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 47"

```
when car issuing for Rental is
    requested (T3/rq)


assess
  justice:
    the performer ... Rental;
    the addressee ... issuer;
 sincerity: <no specific condition>
 truth: Rental is concluded; (T1/ac)
   the driving license of the driver
       of Rental is valid;

   the starting day of Rental is less
        than or equal to Today;

   the number of cars in the car
       group of Rental on Today is
       greater than zero

if complying with the assessment is
    considered justifiable
then
  promise car issuing for Rental [T3/
      pm]
  with the addressee ... Today;
  request car returning for Rental
      [T4/rq]
  with
    the addressee ... Rental;
    the requested ... Rental
else
  decline pick up for Rental [T3/dc]
  with
    the addressee of the decline is
        the driver of Rental;
```

Listing C.29: DB RAC A3/T3 rq Original

```
rule DB RAC A3_T3 rq
when car issuing for Rental is
    requested (T3/rq)
  while rental concluding for Rental
      is accepted (T1/ac)
assess
  justice:
    the performer ... Rental;
    the addressee ... issuer of Car;

  truth:
    the [validity of the driving
        license of the driver] of
        Rental is equal to true;
    the starting day of Rental is
        less than or equal to the
        current day of Today;
    the number of cars in the car
        group of Rental on Today is
        greater than 0
        cars;
if complying with request is
    considered justifiable
then
  promise car issuing for Rental (T3/
      pm)
  with the addressee ... Today;
  request car returning for Rental
      (T4/rq)
  with
    the addressee ... Rental;
    the requested ... Rental;
else
  decline pick up for Rental (T3/dc)
  with
    the addressee of the decline is
        the driver of Rental;
```

Listing C.30: DB RAC A3/T3 rq Adapted

Table C.6: Action Rule DB RAC A3/T3 rq

C.5.6 DB RAC A3/T3 pm

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 48"

```
when car issuing for Rental is
    promised (T3/pm)
  while car returning for Rental is
      promised (T4/pm)

assess
  justice:
    the performer of the promise is
        the car issuer of Rental;
    the addressee of the promise is
        the driver of Rental;
 sincerity: <no specific condition>
 truth:
   there is a Car in the car group of
        Rental on Today;

if complying with the assessment is
    considered justifiable
then
  execute car issuing for Rental [T3/
      ex]
  state car issuing for Rental [T3/st
      ]
  with
    the addressee of the statement is
        the driver of Rental;
    the car of Rental is Car
```

Listing C.31: DB RAC A3/T3 pm Original

```
rule DB RAC A3_T3 pm
when car issuing for Rental is
    promised (T3/pm)
  while car returning for Rental is
      promised (T4/pm)

assess
  justice:
    the performer of the promise is
        the car issuer of Rental;
    the addressee of the promise is
        the driver of Rental;

  truth:
    the car of car group is in the
        Rental of Today;

if complying with promise is
    considered justifiable
then
  execute car issuing for Rental (T3/
      ex)
  state car issuing for Rental (T3/st
      )
  with
    the addressee of the statement is
        the driver of Rental;
    the car of Rental is Car;
```

Listing C.32: DB RAC A3/T3 pm Adapted

Table C.7: Action Rule DB RAC A3/T3 pm

C.5.7 DB RAC A3/T4 st A

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 49"

```
when car returning for Rental is
    stated (T4/st)
  with the actual ... BRANCH
assess
  justice:
    the performer ... Rental;
    the addressee ... Rental;
  sincerity: <no specific condition>
  truth:
    the actual return location of
        Rental is the return location
        of Rental;

    Today is ... Rental

if complying with the assessment is
    considered justifiable
then
  accept car returning ... [T4/ac]
  with the addressee ... Rental;
else
  reject car returning ... [T4/rj]
  with
    the addressee ... Rental;
  request penalty ... Rental [T5/rq]
  with
    the addressee ... Rental;
    the requested ... paying for
        Rental is Now

    the requested penalty ... charge
        of Rental
```

Listing C.33: DB RAC A3/T4 st A Original

```
rule DB RAC A3_T4 st A
when car returning for Rental is
    stated (T4/st)
  with the actual ... BRANCH;
assess
  justice:
    the performer ... Rental;
    the addressee ... Rental;

  truth:
   the actual return location of
       Rental is equal
       to the return location of
       Rental;
    the current day
       of Today is ... Rental;
if complying with state is considered
     justifiable
then
  accept car returning ... (T4/ac)
  with the addressee ... Rental;
else
  reject car returning ... (T4/rj)
  with
    the addressee ... Rental;
  request penalty ... Rental (T5/rq)
  with
    the addressee ... Rental;
    the [requested ... paying] of
        Rental is the current day
        of Now;
    the requested penalty ... charge
        of Rental;
```

Listing C.34: DB RAC A3/T4 st A Adapted

Table C.9: Action Rule DB RAC A3/T4 st A

C.5.8 DB RAC A3/T5 st

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 50"

```
when penalty paying for Rental is
    stated (T5/st)
assess
  justice:
    the performer of the statement is
        the driver of Rental;
    the addressee of the statement is
        the car issuer of Rental;
  sincerity: <no specific condition>
  truth:
    the stated penalty amount of
       Rental is equal to the
       requested penalty amount of
       Rental;
    the stated production time of
       penalty paying for Rental is
       within the promised production
        time of penalty paying for
       Rental

if complying with the assessment is
    considered justifiable
then
  accept penalty paying for Rental
     [T5/ac]
  with
    the addressee of the acceptance
       is the driver of Rental
else
  reject penalty paying for Rental
     [T5/rj]
  with
    the addressee of the acceptance
       is the driver of Rental
```

Listing C.35: DB RAC A3/T5 st Original

```
rule DB RAC A3_T5 st
when penalty paying for Rental is
    stated (T5/st)
assess
  justice:
    the performer of the statement is
        the driver of Rental;
    the addressee of the statement is
        the car issuer of Rental;

  truth:
    the stated penalty amount of
       Rental is equal to the
       requested penalty amount of
       Rental;
    the stated production time of
       penalty paying for Rental is
       less than the promised
       production time of penalty
       paying for Rental;

if complying with state is considered
     justifiable
then
  accept penalty paying for Rental
     (T5/ac)
  with
    the addressee of the acceptance
       is the driver of Rental;
else
  reject penalty paying for Rental
     (T5/rj)
  with
    the addressee of the acceptance
       is the driver of Rental;
```

Listing C.36: DB RAC A3/T5 st Adapted

Table C.10: Action Rule DB RAC A3/T5 st

C.5.9 DB RAC A3/T4 st B

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 51"

```
when car returning for Rental is
    stated (T4/st)
 while penalty paying for Rental is
     accepted (T5/ac)

assess
  justice:
    the performer of the statement is
         the driver of Rental;
    the addressee of the statement is
         the car issuer of Rental;
  sincerity: <no specific condition>
  truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  accept car returning for Rental [T4
      /ac]
  with
    the addressee of the acceptance
       is the driver of Rental
else
  reject car returning for Rental [T4
      /rj]
  with
    the addressee of the rejection is
         the driver of Rental
```

Listing C.37: DB RAC A3/T4 st B Original

```
rule DB RAC A3_T4 st B
when car returning for Rental is
    stated (T4/st)
 while penalty paying for Rental is
     accepted (T5/ac)

assess
  justice:
    the performer of the statement is
         the driver of Rental;
    the addressee of the statement is
         the car issuer of Rental;

if complying with state is considered
     justifiable
then
  accept car returning for Rental (T4
      /ac)
  with
    the addressee of the acceptance
       is the driver of Rental;
else
  reject car returning for Rental (T4
      /rj)
  with
    the addressee of the rejection is
         the driver of Rental;
```

Listing C.38: DB RAC A3/T4 st B Adapted

Table C.11: Action Rule DB RAC A3/T4 st B

C.5.10 DB RAC A6/T6 pm

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 60"

```
when transport completing for
    Transport is promised (T6/pm)
assess
  justice:
    the performer of the promise is
        the transporter for Transport;
    the addressee of the promise is
        the transport manager;
  sincerity: <no specific condition>
  truth:
    it is possible to execute the
        task on day of transport of
        Transport


if complying with the assessment is
    considered justifiable
then
  execute transport completing for
      Transport [T6/ex]
  state transport completing for
      Transport [T6/st]
  with
    the addressee of the statement is
        the transport manager
```

Listing C.39: DB RAC A6/T6 pm Original

```
rule DB RAC A6_T6 pm
when transport completing for
    Transport is promised (T6/pm)
assess
  justice:
    the performer of the promise is
        the transporter of Transport;
    the addressee of the promise is
        the transport manager of
        Transport;

  truth:
    the [possibility of execution of
        the task on day of transport]
        of Transport is equal to
        true;

if complying with promise is
    considered justifiable
then
  execute transport completing for
      Transport (T6/ex)
  state transport completing for
      Transport (T6/st)
  with
    the addressee of the statement is
        the transport manager of
        Transport;
```

Listing C.40: DB RAC A6/T6 pm Adapted

Table C.12: Action Rule DB RAC A6/T6 pm

C.5.11 DB RAC A6/T6 st

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 60"

```
when transport completing for
    Transport is stated (T6/st)
assess
  justice:
    the performer of the statement is
         the transporter for Transport
        ;
    the addressee of the statement is
         the transport manager;
  sincerity: <no specific condition>
  truth:
    the car of Transport has been
        delivered at the
        to-branch of Transport


if complying with the assessment is
    considered justifiable
then
  accept transport completing for
      Transport [T6/ac]
  with
    the addressee of the acceptance
        is the transporter for
        Transport
else
  reject transport completing for
      Transport [T6/rj]
  with
    the addressee of the rejecti is
        the transporter for Transport
```

Listing C.41: DB RAC A6/T6 st Original

```
rule DB RAC A6_T6 st
when transport completing for
    Transport is stated (T6/st)
assess
  justice:
    the performer of the statement is
         the transporter of Transport;
    the addressee of the statement is
         the transport manager of
        Transport;

  truth:
    the [deliverance of car of
        Transport at the
        to_branch] of Transport is
        equal to true;

if complying with state is considered
     justifiable
then
  accept transport completing for
      Transport (T6/ac)
  with
    the addressee of the acceptance
        is the transporter of
        Transport;
else
  reject transport completing for
      Transport (T6/rj)
  with
    the addressee of the rejecti is
        the transporter of Transport;
```

Listing C.42: DB RAC A6/T6 st Adapted

Table C.13: Action Rule DB RAC A6/T6 st

C.5.12 DB RAC A7/T7 rq

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 56"

```
when transport management for Day is
    requested (T7/rq)
assess
  justice:
    the performer of the request is
        the transport manager

    the addressee of the request is
        the transport manager

  sincerity: <no specific condition>
  truth:
    the number of cars to be
        transported on Day is greater
        than zero


if complying with the assessment is
    considered justifiable
then
  promise transport management for
      Day [T7/pm]
  with
    the addressee of the promise is
        the transport manager

else
  decline transport management for
      Day [T7/dc]
  with
    the addressee of the decline is
        the transport manager;

  request transport management for
      Next Day (T7/rq)
```

Listing C.43: DB RAC A7/T7 rq Original

```
rule DB RAC A7_T7 rq
when transport management for Day is
    requested (T7/rq)
assess
  justice:
    the performer of the request is
        the transport manager of
        Transport;
    the addressee of the request is
        the transport manager of
        Transport;

  truth:
    the number of cars to be
        transported in Day is greater
        than 0
        cars;

if complying with request is
    considered justifiable
then
  promise transport management for
      Day (T7/pm)
  with
    the addressee of the promise is
        the transport manager of
        Transport;
else
  decline transport management for
      Day (T7/dc)
  with
    the addressee of the decline is
        the transport manager of
        Transport;
  request transport management for
      Next Day (T7/rq)
```

Listing C.44: DB RAC A7/T7 rq Adapted

Table C.14: Action Rule DB RAC A7/T7 rq

C.5.13 DB RAC A7/T7 pm A

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 57"

```
when transport management for Day is
    promised (T7/pm)

assess
  justice:
    the performer of the promise is
        the transport manager

    the addressee of the promise is
        the transport manager

  sincerity: <no specific condition>
  truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  for each Car in {cars to be
      transported on Day}

    request transport completing for
        Transport (T06/rq)
    with
      the addressee of the request is
          a transporter;

      the car of Transport is Car
```

Listing C.45: DB RAC A7/T7 pm A Original

```
rule DB RAC A7_T7 pm A
when transport management for Day is
    promised (T7/pm)

assess
  justice:
    the performer of the promise is
        the transport manager of
        Transport;
    the addressee of the promise is
        the transport manager of
        Transport;

if complying with promise is
    considered justifiable
then
  for each Car in {the cars to be
      transported of Day}
  {
    request transport completing for
        Transport (T06/rq)
    with
      the addressee of the request is
          the transporter of
          Transport;
      the car of Transport is some
          Car;
  }
```

Listing C.46: DB RAC A7/T7 pm A Adapted

Table C.15: Action Rule DB RAC A7/T7 pm 1

C.5.14 DB RAC A7/T7 pm B

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 57"

```
when transport management for Day is
    promised (T7/pm)
  while
    for each Car in {cars to be
        transported on Day}

      transport completing for
          Transport of Car is accepted
           (T06/ac)


assess
  justice:
    the performer of the promise is
        the transport manager

    the addressee of the promise is
        the transport manager

 sincerity: <no specific condition>
 truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  execute transport management for
      Day [T7/ex]
  state transport management for Day
      [T7/st]
  with
    the addressee of the statement is
         the transport manager
```

Listing C.47: DB RAC A7/T7 pm B Original

```
rule DB RAC A7_T7 pm B
when transport management for Day is
    promised (T7/pm)
  while
    for each Car in {the cars to be
        transported of Day}
    {
      transport completing for
          Transport of Car is accepted
           (T06/ac)
    }

assess
  justice:
    the performer of the promise is
        the transport manager of
        Transport;
    the addressee of the promise is
        the transport manager of
        Transport;


if complying with promise is
    considered justifiable
then
  execute transport management for
      Day (T7/ex)
  state transport management for Day
      (T7/st)
  with
    the addressee of the statement is
         the transport manager of
        Transport;
```

Listing C.48: DB RAC A7/T7 pm B Adapted

Table C.16: Action Rule DB RAC A7/T7 pm 2

C.5.15 DB RAC A7/T7 st

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 58"

```
when transport management for Day is
    stated (T7/st)
assess
  justice:
    the performer of the statement is
         the transport manager

    the addressee of the statement is
         the transport manager

 sincerity: <no specific condition>
 truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  accept transport management for Day
       [T7/ac]
  with
    the addressee of the statement is
         the transport manager

else
  reject transport management for Day
       [T7/rj]
  with
    the addressee of the reject is
        the transport manager
```

Listing C.49: DB RAC A7/T7 st Original

```
rule DB RAC A7_T7 st
when transport management for Day is
    stated (T7/st)
assess
  justice:
    the performer of the statement is
         the transport manager of
        Transport;
    the addressee of the statement is
         the transport manager of
        Transport;

if complying with state is considered
     justifiable
then
  accept transport management for Day
       (T7/ac)
  with
    the addressee of the statement is
         the transport manager of
        Transport;
else
  reject transport management for Day
       (T7/rj)
  with
    the addressee of the reject is
        the transport manager of
        Transport;
```

Listing C.50: DB RAC A7/T7 st Adapted

Table C.17: Action Rule DB RAC A7/T7 st

C.5.16 DB RAC A7/T6 rq

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 59"

```
when transport completing for
    Transport is requested (T6/rq)
  with
    the day ... some day
    the car of Transport is some car
    the from-branch ... branch
    the to-branch ... beanch

assess
  justice:
    the performer of the request is
        the transport manager;

    the addressee of the statement is
         the transporter for Transport
 sincerity: <no specific condition>
 truth: transport is doable


if complying with the assessment is
    considered justifiable
then
  promise transport completing for
      Transport [T6/pm]
  with
    the addressee of the promise is
        the transport manager;

    the promised ... Transport
else
  decline transport completing for
      Transport [T6/dc]
  with
    the addressee of the decline is
        the transport manager
```

Listing C.51: DB RAC A7/T6 rq Original

```
rule DB RAC A7_T6 rq
when transport completing for
    Transport is requested (T6/rq)
  with
    the day ... some day;
    the car of Transport is some car;
    the from_branch ... branch;
    the to_branch ... branch;

assess
  justice:
    the performer of the request is
        the transport manager of
        Transport;
    the addressee of the statement is
         the transporter of Transport;

  truth: the doability
      of transport is equal to true;

if complying with request is
    considered justifiable
then
  promise transport completing for
      Transport (T6/pm)
  with
    the addressee of the promise is
        the transport manager of
        Transport;
    the promised ... Transport;
else
  decline transport completing for
      Transport (T6/dc)
  with
    the addressee of the decline is
        the transport manager of
        Transport;
```

Listing C.52: DB RAC A7/T6 rq Adapted

Table C.18: Action Rule DB RAC A7/T6 rq

C.5.17 DB Volley A1/T1 rq

This example has been taken from "DEMO Bachelor+ 3.5 M3 p slide 20" This example does contain the common typos and the formulation error of the Date Time condition.

```
when membership starting for new
    Membership is requested (T1/rq)
  with
    the member of Membership is a
        Person
    the starting day of Membership is
         a Day

assess
  justice:
    the performer of the request is
        the member of Membership
  sincerity: <no specific condition>
  truth:
    Day is the first day of some
        Month;
    Month is greater than Current
        Month;
    the age of Person is equal to or
        greater than the minimal age
        in the year of Day;
    the number of members on Day is
        less than the max members in
        the year of Day

if complying with the assessment is
    considered justifiable
then
  promise membership starting for
      Membership [T1/pm]
else
  decline membership starting for
      Membership [T1/dc]
```

Listing C.53: DB Volley A1/T1 rq Original

```
rule DB Volley A1_T1 rq
when membership starting for new
    Membership is requested (T1/rq)
  with
    the member of Membership is some
        Person;
    the starting day of Membership is
         some Day;

assess
  justice:
    the performer of the request is
        the member of Membership;

  truth:
    the Day of Date is equal to the
        first day of some Month;
    the Month of Date is greater than
         the Current Month of Date;
    the age of Person is equal to or
        greater than the minimal age
        in the year of Day;
    the number of members of Day is
        less than the max members in
        the year of Day;

if complying with request is
    considered justifiable
then
  promise membership starting for
      Membership (T1/pm)
else
  decline membership starting for
      Membership (T1/dc)
```

Listing C.54: DB Volley A1/T1 rq Adapted

Table C.19: Action Rule DB Volley A1/T1 rq

C.5.18 DB Volley A1/T1 pm

This example has been taken from "TEOO 3rd ed fig 4.12 page 62". This example complies with the new grammar. It contains the common typos including the assessment keyword.

```
when membership start for Membership
    is promised (T1/pm)
assess
  justice:
    the performer of the promise is
        the membership starter of
        Membership

        the addressee of the promise
            is the member of
            Membership;
  sincerity: <no specific condition>
  truth: <no specific condition>
if complying with the assessment is
    considered justifiable
then
  request membership payment for
      Membership [T2/rq]
  with
    the addressee of the request is
        the payer of Membership;
    the amount to pay of Membership
        is equal to the first fee of
        Membership
```

Listing C.55: DB Volley A1/T1 pm Original

```
rule DB Volley A1_T1 pm
when membership start for Membership
    is promised (T1/pm)
assess
  justice:
    the performer of the promise is
        the membership starter of
        Membership;

    the addressee of the promise is
        the member of Membership;


if complying with request is
    considered justifiable
then
  request membership payment for
      Membership (T2/rq)
  with
    the addressee of the request is
        the payer of Membership;
    the amount to pay of Membership
        is the first fee of Membership
        ;
```

Listing C.56: DB Volley A1/T1 pm Adapted

Table C.20: Action Rule DB Volley A1/T1 rq

C.5.19 DM RAC A1/T1 new rq

This example has been taken from "master slide 3.4+ module 2 p slide 11 and 12"

```
when rental concluding for new Rental
     is requested (T1/rq)
  with
    the starting day ... DAY
    the ending day ... DAY
    the renter ... PERSON
    the driver ... PERSON
    the payer ... PERSON
    the car group ... CAR GROUP
    the pick-up location ... BRANCH
    the drop-off location ... BRANCH
assess
  justice:
    the performer of the request is
      the renter of Rental;
    the addressee of the request is
       some rental concluder;
  sincerity: <no specific condition>
  truth:
    the starting day ... Rental;
    the ending day ... Rental;
    the ending day ... Rental;
    the duration of Rental is less
       than or equal to the max
       rental duration in the year of
        the starting day of Rental;
    the number of cars in the car
       group of Renta on every day in
        the rental period of Rental
       is greater than zero
```

Listing C.57: DM RAC A1/T1 new rq Original

```
rule [DM RAC A1_T1 new rq]
when rental concluding for new Rental
     is requested (T1/rq)
  with
    the starting day ... day;
    the ending day ... day;
    the renter ... person;
    the driver ... person;
    the payer ... person;
    the car group ... car group;
    the pick_up location ... branch;
    the drop_off location ... branch;
assess
  justice:
    the performer of the request is
       the renter of Rental;
    the addressee of the request is
       some rental concluder;

  truth:
    the starting day ... Rental;
    the ending day ... Rental;
    the ending day ... Rental;
    the duration of Rental is less
       than or equal to
    the max rental duration in the
       year of the starting day of
       Rental;
    the number of cars in the car
       group of Rental on every day
       in the rental period of Rental
        is greater than 0
       cars;
```

Listing C.58: DM RAC A1/T1 new rq Adapted

Table C.21: Action Rule DM RAC A1/T1 new rq (event and assess)

```
if complying with request is
    considered justifiable
  then
    request rental payment for Rental
        [T2/rq]
    with the addressee of the request
        is the payer of Rental;
      the requested production day of
          rental payment for Rental
          is less than or equal
          to the starting day of
          Rental;
      the requested paid rental
          amount of Rental is equal
          to the rental charge of
          Rental;
else
  decline rental concluding for
      Rental [T1/dc]
  with
    the addressee of the decline is
        the renter of Rental
```

Listing C.59: DM RAC A1/T1 new rq Original

```
if complying     with request is
    considered justifiable
then
  request rental payment for Rental
      (T2/rq)
  with
    the addressee of the request is
        the payer of Rental;
    the requested production day of
        rental payment for Rental is
        the starting day of Rental;
    the requested paid rental amount
        of Rental is the rental charge
         of Rental;
else
  decline rental concluding for
      Rental (T1/dc)
  with
    the addressee of the decline is
        the renter of Rental;
```

Listing C.60: DM RAC A1/T1 new rq Adapted

Table C.22: Action Rule DM RAC A1/T1 new rq (result)

C.5.20 DM RAC A1/T1 rq

This example has been taken from "master slide 3.4+ module 2 p slide 13"

```
when rental concluding for Rental is
    requested (T1/rq)
 while rental payment for Rental is
     promised (T2/pm)

assess
  justice:
    the performer ... of Rental;
    the addressee ... of Rental;
  sincerity: <no specific condition>
  truth:
    the promised paid ... Rental;
    the promised production day of
        rental payment for Rental is
        less than or equal to the
        starting day of Rental

if complying with request is
    considered justifiable
then
  promise rental concluding for
      Rental [T1/pm]
  with
    the addressee of the promise is
        the rental concluder of Rental
else
  decline rental concluding for
      Rental [T1/dc]
  with
    the addressee of the decline is
        the renter of Rental
```

Listing C.61: DM RAC A1/T1 rq Original

```
rule [DM RAC A1_T1 rq]
when rental concluding for Rental is
    requested (T1/rq)
 while rental payment for Rental is
     promised (T2/pm)

assess
  justice:
    the performer ... of Rental;
    the addressee ... of Rental;

  truth:
    the promised paid ... Rental;
    the promised production day of
        rental payment of Rental is
        less than or equal to the
        starting day of Rental;

if complying with request is
    considered justifiable
then
  promise rental concluding for
      Rental (T1/pm)
  with
    the addressee of the promise is
        the rental concluder of Rental
        ;
else
  decline rental concluding for
      Rental (T1/dc)
  with
    the addressee of the decline is
        the renter of Rental;
```

Listing C.62: DM RAC A1/T1 rq Adapted

Table C.23: Action Rule DM RAC A1/T1 rq

C.5.21 DM RAC A1/T2 st (sl14)

This example has been taken from "master slide 3.4+ module 2 p slide 14"

```
when rental payment for Rental is
    stated (T2/st)

assess
  justice:
    the performer of the statement is
        the renter of Rental;
    the addressee of the statement is
        the rental concluder of
       Rental;
  sincerity:  <no specific condition>

  truth:
    the stated paid ... Rental

if complying with statement is
    considered justifiable
then
  accept rental payment for Rental
     [T2/ac]
  with
    the addressee of the acceptance
       is the renter of Rental
else
  reject rental payment for Rental
     [T2/rj]
  with
    the addressee of the rejection is
        the renter of Rental
```

Listing C.63: DM RAC A1/T2 st (sl14) Original

```
rule [DM RAC A1_T2 st14]
when rental payment for Rental is
    stated (T2/st)

assess
  justice:
    the performer of the statement is
        the renter of Rental;
    the addressee of the statement is
        the rental concluder of
       Rental;

  truth:
    the stated paid rental amount of
       Rental is equal to the
       promised paid rental amount of
        Rental;
    the stated ... Rental;

if complying with state is considered
    justifiable
then
  accept rental payment for Rental
     (T2/ac)
  with
    the addressee of the acceptance
       is the renter of Rental;
else
  reject rental payment for Rental
     (T2/rj)
  with
    the addressee of the rejection is
        the renter of Rental;
```

Listing C.64: DM RAC A1/T2 st (sl14) Adapted

Table C.24: Action Rule DM RAC A1/T2 st

C.5.22 DM RAC A1/T1 pm

This example has been taken from "master slide 3.4+ module 2 p slide 13"

```
when rental concluding for Rental is
    promised (T1/pm)
  while rental payment for Rental is
      accepted (T2/ac)

assess
  justice:
    the performer of the promise is
        the rental concluder of Rental
        ;
    the addressee of the promise is
        the renter of Rental;
  sincerity: <no specific condition>
  truth:  <no specific condition>

if complying with promise is
    considered justifiable
then
  execute rental concluding for
      Rental [T1/ex]
  state rental concluding for Rental
      [T1/st]
  with
    the addressee of the statement is
        the renter of Rental
```

Listing C.65: DM RAC A1/T1 pm Original

```
rule [DM RAC A1_T1 pm]
when rental concluding for Rental is
    promised (T1/pm)
  while rental payment for Rental is
      accepted (T2/ac)

assess
  justice:
    the performer of the promise is
        the rental concluder of Rental
        ;
    the addressee of the promise is
        the renter of Rental;


if complying with promise is
    considered justifiable
then
  execute rental concluding for
      Rental (T1/ex)
  state rental concluding for Rental
      (T1/st)
  with
    the addressee of the statement is
        the renter of Rental;
```

Listing C.66: DM RAC A1/T1 pm Adapted

Table C.25: Action Rule DM RAC A1/T1 pm

C.5.23 DM RAC A1/T1 new rq

This example has been taken from "master slide 3.4+ module 3 p slide 39, 41"

```
when rental concluding for new Rental
    is requested (T1/rq)
  with
    the starting day ... day
    the ending day ... day
    the renter of ... person
    the driver of ... person
    the payer of ... person
    the car group of Rental is some
       car group
    the pick-up location of Rental is
        some branch
    the drop-off location of Rental
       is some branch
assess
  justice:
    the performer of the request is
       the renter of Rental;
    the addressee of the request is
       a rental concluder;
  sincerity: <no specific condition>
  truth:
    the starting ... Rental;
    the ending ... Rental;
    the ending ... Rental;
    the duration of Rental is less
       than or equal to
    the max rental duration in the
       year of the starting day of
       Rental;
    the number of cars ... than zero
```

Listing C.67: DM RAC A1/T1 new rq Original

Listing C.68: DM RAC A1/T1 new rq Adapted

Table C.26: Action Rule DM RAC A1/T1 new rq (event and assess)

```
if complying     with the assessment
    is considered justifiable
then
  request rental payment for Rental
      [T2/rq]
  with
    the addressee of the request is
        the payer of Rental;
    the requested production day of
        rental payment for Rental is
        less than or equal
        to the starting day of Rental;

    the requested paid rental amount
        of Rental is equal
        to the rental charge of Rental
        ;
else
  decline rental concluding for
      Rental [T1/dc]
  with
    the addressee of the decline is
        the renter of Rental
```

Listing C.69: DM RAC A1/T1 new rq Original

Listing C.70: DM RAC A1/T1 new rq Adapted

Table C.27: Action Rule DM RAC A1/T1 new rq (result)

C.5.24 DM RAC A1/T1 rq

This example has been taken from "master slide 3.4+ module 3 p slide 44"

```
when rental concluding for Rental is
    requested (T1/rq)
 while rental payment for Rental is
     promised (T2/pm)

assess
  justice:
    the performer of the request is
        the renter of Rental;
    the addressee of the request is
        the rental concluder of Rental
        ;
  sincerity: <no specific condition>
  truth:
    the promised ... Rental;
    the promised production ...
        Rental

if complying with the assessment is
    considered justifiable
then
  promise rental concluding for
      Rental [T1/pm]
  with
    the addressee of the promise is
        the rental concluder of Rental
else
  decline rental concluding for
      Rental [T1/dc]
  with
    the addressee of the decline is
        the renter of Rental
```

Listing C.71: DM RAC A1/T1 rq Original

```
rule [DM RAC A1_T1 rq]
when rental concluding for Rental is
    requested (T1/rq)
 while rental payment for Rental is
     promised (T2/pm)

assess
  justice:
    the performer of the request is
        the renter of Rental;
    the addressee of the request is
        the rental concluder of Rental
        ;

  truth:
    the promised ... Rental;
    the promised production ...
        Rental;

if complying with request is
    considered justifiable
then
  promise rental concluding for
      Rental (T1/pm)
  with
    the addressee of the promise is
        the rental concluder of Rental
        ;
else
  decline rental concluding for
      Rental (T1/dc)
  with
    the addressee of the decline is
        the renter of Rental;
```

Listing C.72: DM RAC A1/T1 rq Adapted

Table C.28: Action Rule DM RAC A1/T1 rq

C.5.25 DM RAC A1/T2 st (sl46)

This example has been taken from "master slide 3.4+ module 3 p slide 46"

```
when rental payment for Rental is
    stated (T2/st)

assess
  justice:
    the performer of the statement is
         the renter of Rental;
    the addressee of the statement is
         the rental concluder of
        Rental;
 sincerity: <no specific condition>
 truth:
   the stated paid rental ... Rental;
   the stated production ... Rental

if complying with the assessment is
    considered justifiable
then
  accept rental payment for Rental
      [T2/ac]
  with
    the addressee of the acceptance
        is the renter of Rental
else
  reject rental payment for Rental
      [T2/rj]
  with
    the addressee of the rejection is
         the renter of Rental
```

Listing C.73: DM RAC A1/T2 st (sl46) Original

```
rule [DM RAC A1_T2 st46]
when rental payment for Rental is
    stated (T2/st)

assess
  justice:
    the performer of the statement is
         the renter of Rental;
    the addressee of the statement is
         the rental concluder of
        Rental;

  truth:
    the stated paid ... Rental;
    the stated production ... Rental;

if complying with state is considered
     justifiable
then
  accept rental payment for Rental
      (T2/ac)
  with
    the addressee of the acceptance
        is the renter of Rental;
else
  reject rental payment for Rental
      (T2/rj)
  with
    the addressee of the rejection is
         the renter of Rental;
```

Listing C.74: DM RAC A1/T2 st (sl46) Adapted

Table C.29: Action Rule DM RAC A1/T2 st

C.5.26 DM RAC A1/T1 pm

This example has been taken from "master slide 3.4+ module 3 p slide 48"

```
when rental concluding for Rental is
    promised (T1/pm)
  while rental payment for Rental is
      accepted (T2/ac)

assess
  justice:
    the performer of the promise is
        the rental concluder of Rental
        ;
    the addressee of the promise is
        the renter of Rental;
 sincerity: <no specific condition>
 truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  execute rental concluding for
      Rental [T1/ex]
  state rental concluding for Rental
      [T1/st]
  with
    the addressee of the statement is
        the renter of Rental
```

Listing C.75: DM RAC A1/T1 pm Original

```
rule [DM RAC A1_T1 pm]
when rental concluding for Rental is
    promised (T1/pm)
  while rental payment for Rental is
      accepted (T2/ac)

assess
  justice:
    the performer of the promise is
        the rental concluder of Rental
        ;
    the addressee of the promise is
        the renter of Rental;



if complying with promise is
    considered justifiable
then
  execute rental concluding for
      Rental (T1/ex)
  state rental concluding for Rental
      (T1/st)
  with
    the addressee of the statement is
        the renter of Rental;
```

Listing C.76: DM RAC A1/T1 pm Adapted

Table C.30: Action Rule DM RAC A1/T1 pm

C.5.27 DM Pizza A1/T1 new rq

This example has been taken from "master slide 3.4+ module 3 p slide 53"

```
when order completion for new Order
    is requested (T01/rq)
  with
    the customer of Order ... person
    the contents of Order is a set of
         order item
  with
    the pizza kind ... pizza kind
    the amount of Order ... number
    the delivery time ... time

assess
  justice:
    the performer ... Order
    the addressee ... completer
  sincerity: <no specific condition>
  truth:
    for each Order Item in the
        contents of Order:
      the pizza kind of Order Item is
          available

        the amount of Order Item is
            producible

      the requested production ...
          than Now plus 10
          minutes

if complying with the assessment is
    considered justifiable
then promise ... Order [T01/pm]
  with the addressee ... of Order
else decline ... Order [T01/dc]
  with the addressee ... of Order
```

Listing C.77: DM Pizza A1/T1 new rq Original

```
rule [DM Pizza A1_T1 new rq]
when order completion for new Order
    is requested (T01/rq)
  with
    the customer of Order ... person;
    the contents of Order is set of
        order item {
    with
      the pizza kind ... pizza kind;
      the amount of Order ... number;
      the delivery time ... time;
    };
assess
  justice:
    the performer ... Order;
    the addressee ... completer;

  truth:
    for each Order Item in {the
        contents of Order} {
      the availability of pizza kind
          of Order Item is equal to
          true;
      the [producebility of amount]
          of Order Item is equal to
          true;
      the requested production ...
          than the [Now plus10
          minutes] of Time;
    }
if complying with request is
    considered justifiable
then promise ... Order (T01/pm)
  with the addressee ... of Order;
else decline ... Order (T01/dc)
  with the addressee ... of Order;
```

Listing C.78: DM Pizza A1/T1 new rq Adapted

Table C.31: Action Rule DM Pizza A1/T1 new rq

C.5.28 DM Pizza A1/T1 pm A

This example has been taken from "master slide 3.4+ module 3 p slide 55"

```
when order completion for Order is
    promised (T01/pm)
assess
  justice:
    the performer of the promise is
        the order completer of Order
    the addressee of the promise is
        the order completer of Order
  sincerity: <no specific condition>
  truth:  <no specific condition>
if complying with the assessment is
    considered justifiable
then
  request order payment for Order
      [T02/rq]
  with
    the addressee of the request is
        the customer of Order
    the amount to pay of Order is
        total price of Order
```

Listing C.79: DM Pizza A1/T1 pm A Original

```
rule [DM Pizza A1_T1 pm A]
when order completion for Order is
    promised (T01/pm)
assess
  justice:
    the performer of the promise is
        the order completer of Order;
    the addressee of the promise is
        the order completer of Order;


if complying with promise is
    considered justifiable
then
  request order payment for Order
      (T02/rq)
  with
    the addressee of the request is
        the customer of Order;
    the amount to pay of Order is
        total price of Order;
```

Listing C.80: DM Pizza A1/T1 pm A Adapted

Table C.32: Action Rule DM Pizza A1/T1 pm A

C.5.29 DM Pizza A1/T1 pm B

This example has been taken from "master slide 3.4+ module 3 p slide 57"

```
when order completion for Order
is promised (T01/pm)
  while order payment for Order is
      promised (T02/pm)

assess
  justice:
    the performer of the promise is
        the order completer of Order
    the addressee of the promise is
        the order completer of Order
  sincerity: <no specific condition>
  truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  request order baking for Order [T03
      /rq]
  with
    the addressee of the request is
        some order baker
```

Listing C.81: DM Pizza A1/T1 pm B Original

```
rule [DM Pizza A1_T1 pm B]
when order completion for Order is
    promised (T01/pm)
  while order payment for Order is
      promised (T02/pm)

assess
  justice:
    the performer of the promise is
        the order completer of Order;
    the addressee of the promise is
        the order completer of Order;

if complying with promise is
    considered justifiable
then
  request order baking for Order (T03
      /rq)
  with
    the addressee of the request is
        some order baker;
```

Listing C.82: DM Pizza A1/T1 pm B Adapted

Table C.33: Action Rule DM Pizza A1/T1 pm B

C.5.30 DM Pizza A1/T2 st

This example has been taken from "master slide 3.4+ module 3 p slide 58"

```
when order payment for Order is
    stated (T02/st)

assess
  justice:
    the performer of the statement is
        the customer of Order
    the addressee of the statement is
        the order completer of Order
  sincerity: <no specific condition>
  truth:
    the stated paid amount of Order
        is equal to the promised
        amount to pay of Order
    the stated production time of
        order payment for Order is
        equal to the promised
        production time of order
        payment for Order

if complying with the assessment is
    considered justifiable
then
  accept order payment for Order [T02
      /ac]
  with
    the addressee of the acceptance
        is the customer of Order
else
  reject order payment for Order [T02
      /rj]
  with
    the addressee of the rejection is
        the customer of Order
```

Listing C.83: DM Pizza A1/T2 st Original

```
rule [DM Pizza A1_T2 st]
when order payment for Order is
    stated (T02/st)

assess
  justice:
    the performer of the statement is
        the customer of Order;
    the addressee of the statement is
        the order completer of Order;

  truth:
    the stated paid amount of Order
        is equal to the promised
        amount to pay of Order;
    the [stated production time of
        order payment] of Order is
        equal to the [promised
        production time of order
        payment] of Order;

if complying with state is considered
     justifiable
then
  accept order payment for Order (T02
      /ac)
  with
    the addressee of the acceptance
        is the customer of Order;
else
  reject order payment for Order (T02
      /rj)
  with
    the addressee of the rejection is
        the customer of Order;
```

Listing C.84: DM Pizza A1/T2 st Adapted

Table C.34: Action Rule DM Pizza A1/T2 st

C.5.31 DM Pizza A1/T3 st

This example has been taken from "master slide 3.4+ module 3 p slide 59"

```
when order baking for Order is stated
     (T03/st)

assess
  justice:
    the performer of the statement is
         the order baker of Order
  the addressee of the statement is
      the order completer of Order
  sincerity: <no specific condition>
  truth:
    the stated contents of Order is
        equal to the promised contents
         of Order
    the stated production time of
        order baking for Order is
        equal to the promised
        production time of order
        baking for Order

if complying with the assessment is
    considered justifiable
then
  accept order baking for Order [T03/
      ac]
  with
    the addressee of the acceptance
        is the order baker of Order
else
  reject order baking for Order [T03/
      rj]
  with
    the addressee of the rejection is
         the order baker of Order
```

Listing C.85: DM Pizza A1/T3 st Original

```
rule [DM Pizza A1_T3 st]
when order baking for Order is stated
     (T03/st)

assess
  justice:
    the performer of the statement is
         the order baker of Order;
    the addressee of the statement is
         the order completer of Order;

  truth:
    the stated contents of Order is
        equal to the promised contents
         of Order;
    the [stated production time of
        order baking] of Order is
        equal to the [promised
        production time of order
        baking] of Order;

if complying with state is considered
     justifiable
then
  accept order baking for Order (T03/
      ac)
  with
    the addressee of the acceptance
        is the order baker of Order;
else
  reject order baking for Order (T03/
      rj)
  with
    the addressee of the rejection is
         the order baker of Order;
```

Listing C.86: DM Pizza A1/T3 st Adapted

Table C.35: Action Rule DM Pizza A1/T3 st

C.5.32 DM Pizza A1/T1 pm

This example has been taken from "master slide 3.4+ module 3 p slide 61"

```
when order completion for Order is
    promised (T01/pm)
  while
    order payment for Order is
        accepted (T02/ac)
    order baking for Order is
        accepted (T03/ac)

assess
  justice:
    the performer of the promise is
        the order completer of Order
    the addressee of the promise is
        the order completer of Order
  sincerity: <no specific condition>
  truth:  <no specific condition>

if complying with the assessment is
    considered justifiable
then
  execute order completion for Order
      [T01/ex]
  state order completion for Order
      [T01/st]
  with
    the addressee of the statement is
        the customer of Order
```

Listing C.87: DM Pizza A1/T1 pm Original

```
rule [DM Pizza A1_T1 pm]
when order completion for Order is
    promised (T01/pm)
  while
    order payment for Order is
        accepted (T02/ac)
    order baking for Order is
        accepted (T03/ac)

assess
  justice:
    the performer of the promise is
        the order completer of Order;
    the addressee of the promise is
        the order completer of Order;

if complying with promise is
    considered justifiable
then
  execute order completion for Order
      (T01/ex)
  state order completion for Order
      (T01/st)
  with
    the addressee of the statement is
        the customer of Order;
```

Listing C.88: DM Pizza A1/T1 pm Adapted

Table C.36: Action Rule DM Pizza A1/T1 pm

Appendix D

Exchange Metamodel

D.1 Model base

```
<xs:schema id="DEMO3"
    targetNamespace="http://www.ee-institute.org/DEMOProp"
    elementFormDefault="qualified"
    xmlns="http://www.ee-institute.org/DEMOProp"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
```

Listing D.1: Schema specification

```
 <xs:element name="DEMOmodel">
   <xs:annotation>
     <xs:documentation xml:lang="en">
       This XSD schema is the DEMO 3.8 schema to exchange DEMO models.
       Each element is documented and restrictedto the specifications that
           find origin in DEMOSL3.7 and DEMOBAKER.
       The distribution file has a filename with a version number yyyymmdd.
           The date of release.
       All relevant documentation per element is added in Annotations, not in
           comments.
       The DEMOmodel must comply to this XSD, but still can contain other
           errors that cannot be forced with XSD.
       The validation of the complete model is also available for public use.
     </xs:documentation>
   </xs:annotation>
```

Listing D.2: Model specification

```
     <xs:attribute name="Name" type="DEMOModelName" use="optional"/>
```

Listing D.3: Model attributes specification

D.2 Core collections

```
<xs:element name="TransactionKinds" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      A listing of all available Transaction Kinds in this model.
      Transaction Kinds are abbreviated with TK.
      Not all TKs in this list have to be used in diagrams or
          connections, but will give a warning upon validation.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TransactionKind" type="TransactionKind"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Transaction Kind element can be added. When the
                TransactionKinds is present,
            a minimum of one TransactionKind must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.4: Model transactionkind collection

```
<xs:element name="ElementaryActorRoles" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      A listing of all available Elementary Actor Roles in this model.
      Elementary Actor Roles are abbreviated with EAR.
      Not all EARs in this list have to be used in diagrams or
          connections, but will give a warning upon validation.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ElementaryActorRole" type="ElementaryActorRole"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Elementary Actor Role element can be added. When the
                ElementaryActorRoles is present,
            a minimum of one ElementaryActorRole must be added.
```

```
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.5: Model elementary actor role collection

```
<xs:element name="AggregateTransactionKinds" minOccurs="0" maxOccurs="1
    ">
  <xs:annotation>
    <xs:documentation>A listing of all available Aggregate Transaction
        Kinds</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="AggregateTransactionKind" type="
          AggregateTransactionKind" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Aggregate Transaction Kind element can be added. When the
                AggregateTransactionKinds is present,
            a minimum of one AggregateTransactionKind must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.6: Model aggregate transactionkind collection

```
<xs:element name="CompositeActorRoles" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Composite Actor Roles</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CompositeActorRole" type="CompositeActorRole"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Composite Actor Role element can be added. When the
                CompositeActorRoles is present,
            a minimum of one CompositeActorRole must be added.
          </xs:documentation>
```

```
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.7: Model composite actor role collection

```
<xs:element name="Actors" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Actors</
       xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Actor" type="Actor" minOccurs="1" maxOccurs="
         unbounded">
        <xs:annotation>
          <xs:documentation>
            A Actor element can be added. When the Actor is present,
            a minimum of one Actor must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.8: Model actor collection

```
<xs:element name="IndependentFactKinds" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Independent Fact Kinds<
       /xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="IndependentFactKind" type="IndependentFactKind"
         minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Independent Fact Kind element can be added. When the
               IndependentFactKinds is present,
            a minimum of one IndependentFactKind must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
```

Listing D.9: Model independent fact kind collection

```
<xs:element name="TransactionProcessStepKinds" minOccurs="0" maxOccurs=
    "1">
  <xs:annotation>
    <xs:documentation>A listing of all available Transaction Process
        Step Kinds</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TransactionProcessStepKind" type="
          TransactionProcessStepKind" minOccurs="1" maxOccurs="unbounded"
          >
        <xs:annotation>
          <xs:documentation>
            A Transaction Process Step Kind element can be added. When
                the TransactionProcessStepKinds is present,
            a minimum of one TransactionProcessStepKind must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.10: Model transaction proces step kind collection

```
<xs:element name="AttributeTypes" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available AttributeTypes</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="AttributeType" type="AttributeType" minOccurs="1
          " maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Attribute Type element can be added. When the
                AttributeTypes is present,
            a minimum of one AttributeType must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
```

Listing D.11: Model attribute collection

```
<xs:element name="EntityTypes" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available EntityTypes</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EntityType" type="EntityType" minOccurs="1"
          maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Entity Type element can be added. When the EntityTypes is
                present,
            a minimum of one EntityType must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.12: Model entity type collection

```
<xs:element name="ActionRuleTypes" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Action Rule Types</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ActionRuleType" type="ActionRuleType" minOccurs=
          "0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Action Rule Type element can be added. When the
                ActionRuleTypes is present,
            a minimum of one ActionRuleType must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.13: Model action rule collection

D.3 Connection collections

```
<xs:element name="Connections" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available connections</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Connection" type="Connection" minOccurs="1"
          maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Connection element can be added. When the Connections is
                present,
            a minimum of one Connection must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.14: Model connection collection

D.4 Core diagram collections

```
<xs:element name="OrganisationConstructionDiagrams" minOccurs="0"
    maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Organisation
        Construction Diagrams</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OrganisationConstructionDiagram" type="
          OrganisationConstructionDiagram" minOccurs="1" maxOccurs="
          unbounded">
        <xs:annotation>
          <xs:documentation>
```

```
            An Organisation Construction Diagram element can be added.
                When the OrganisationConstructionDiagrams is present,
            a minimum of one OrganisationConstructionDiagram must be
                added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.15: Model construction diagram collection

```
<xs:element name="ProcessStructureDiagrams" minOccurs="0" maxOccurs="1"
    >
  <xs:annotation>
    <xs:documentation>A listing of all available Process Structure
        Diagrams</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ProcessStructureDiagram" type="
          ProcessStructureDiagram" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            An Process Structure Diagram element can be added. When the
                ProcessStructureDiagrams is present,
            a minimum of one ProcessStructureDiagram must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.16: Model process diagram collection

```
<xs:element name="ObjectFactDiagrams" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Object Fact Diagrams</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ObjectFactDiagram" type="ObjectFactDiagram"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
```

```
            An Object Fact Diagram element can be added. When the
                ObjectFactDiagrams is present,
            a minimum of one ObjectFactDiagram must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.17: Model fact diagram collection

```
<xs:element name="ActorFunctionDiagrams" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Actor Function Diagrams
        </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ActorFunctionDiagram" type="ActorFunctionDiagram
          " minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            An Actor Function Diagram element can be added. When the
                ActorFunctionDiagrams is present,
            a minimum of one ActorFunctionDiagram must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.18: Model actor function diagram collection

```
<xs:element name="TransactionProcessDiagrams" minOccurs="0" maxOccurs="
    1">
  <xs:annotation>
    <xs:documentation>A listing of all available Transaction Process
        Diagrams</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TransactionProcessDiagram" type="
          TransactionProcessDiagram" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
```

```
            An Transaction Process Diagram element can be added. When the
                 TransactionProcessDiagrams is present,
            a minimum of one TransactionProcessDiagram must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.19: Model transaction diagram collection

```
<xs:element name="ActionRuleDiagrams" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>A listing of all available Action Rule Diagrams</
        xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ActionRuleDiagram" type="ActionRuleDiagram"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            An Action Rule Diagram element can be added. When the
                ActionRuleDiagrams is present,
            a minimum of one ActionRuleDiagram must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.20: Model action rule diagram collection

D.5 Core validation collection

```
<xs:element name="ValidationResults" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      A listing of all Errors in this model.
      This is beyond the DEMO meta model but is used for communication
          about the errors found in the model
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
```

```
        <xs:sequence>
          <xs:element name="ValidationResult" type="ValidationResult"
              minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                An Validation Result element can be added. When the
                    ValidationResults is present,
                a minimum of one ValidationResult must be added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

Listing D.21: Model validation result collection

D.6 Basic data types

```
<xs:simpleType name="PositiveInteger" id="PositiveInteger">
  <xs:annotation>
    <xs:documentation>
      PositiveInteger allows for an integer from 1 to infinity.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:int">
    <xs:minInclusive value="1"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.22: Positive integer

```
<xs:simpleType name="PositiveDouble" id="PositiveDouble">
  <xs:annotation>
    <xs:documentation>
      PositiveDouble allows for an double from 0 to infinity.
      0 is not included, it is not positive.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:double">
    <xs:minExclusive value="0"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.23: Positive integer

```
<xs:complexType name="Location">
  <xs:annotation>
    <xs:documentation>
      Location is the x,y coordinate and location Type of the location of
      an diagram element on a diagram.
      The location complexType is used for lines and diagram Elements.
      To compute the center of a diagram element,
      use CenterX = Left + Width/2; CenterY = Top + Height/2;
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Top" type="xs:double">
      <xs:annotation>
        <xs:documentation>
          Top is the y coordinate (vertical) of the location of an diagram
              element
          on a diagram. Positive y is upward.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Left" type="xs:double">
      <xs:annotation>
        <xs:documentation>
          Left is the x coordinate (horizontal) of the location of an diagram
              element
          on a diagram. Postive x is to the right.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Zorder" type="xs:integer">
      <xs:annotation>
        <xs:documentation>
          Zorder is the order of drawing the elements on a diagram.
          The lowest element has the smallest Zorder. In case of equal numbers
               the result is undefined.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Type" type="xs:string" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          The Type is usable for graphical applications to store their
              specific properties on a location.
          This field is not restricted nor controlled by this standard.
        </xs:documentation>
      </xs:annotation>
```

```
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.24: Location type

```
<xs:complexType name="Size">
  <xs:annotation>
    <xs:documentation>
      Size is the complexType that stores the dimensions of an diagram
          element.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Width" type="PositiveDouble">
      <xs:annotation>
        <xs:documentation>
          Width is the x coordinate difference of an diagram element.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Height" type="PositiveDouble">
      <xs:annotation>
        <xs:documentation>
          Height is the y coordinate difference of an diagram element.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.25: Size type

```
<xs:complexType name="Line">
  <xs:annotation>
    <xs:documentation>
      Line is the visualisation concept for a connection on a diagram.
      Using lines you can show the preferred path of the line on the diagram.
      The coordinates follow the same definitions as given in Location.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Start" type="Location">
      <xs:annotation>
        <xs:documentation>
          Start is the starting location of a line between two diagram
              elements.
          This does not represent the connection of the elements.
```

```
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Midpoints" type="Location" minOccurs="0" maxOccurs="
        unbounded">
      <xs:annotation>
        <xs:documentation>
          Midpoints is the set of locations of a line between two diagram
              elements, more that start and end.
          This does not represent the connection of the elements.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="End" type="Location">
      <xs:annotation>
        <xs:documentation>
          End is the end location of a line between two diagram elements.
          This does not represent the connection of the elements.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Type" type="xs:string">
      <xs:annotation>
        <xs:documentation>
          The Type is usable for graphical applications to store their
              specific properties on a line or line segment.
          This field is not restricted nor controlled by this standard.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.26: Line type

D.7 Elements

```
<xs:complexType name="TransactionKind">
  <xs:sequence>
    <xs:element name="Identification" type="TransactionKindId"></xs:element>
    <xs:element name="Name" type="TransactionKindName"></xs:element>
    <xs:element name="TransactionSort" type="TransactionSort" default="
        unknown"></xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="TransactionKindGuid" use="required"/>
</xs:complexType>
```

Listing D.27: Transaction kind

This exchange rule set described in XSD can sufficiently restrict field content for each Transaction Kind (TK).

```
<xs:complexType name="AggregateTransactionKind">
  <xs:sequence>
    <xs:element name="Identification" type="AggregateTransactionKindId"></
       xs:element>
    <xs:element name="Name" type="TransactionKindName"></xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="AggregateTransactionKindGuid" use="required"/
     >
</xs:complexType>
```

Listing D.28: Aggregate transaction kind

```
<xs:complexType name="ElementaryActorRole">
  <xs:sequence>
    <xs:element name="Identification" type="ActorRoleId"></xs:element>
    <xs:element name="Name" type="ActorRoleName"></xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="ElementaryActorRoleGuid" use="required"/>
</xs:complexType>
```

Listing D.29: Elementary actor role

```
<xs:complexType name="CompositeActorRole">
  <xs:sequence>
    <xs:element name="Identification" type="CompositeActorRoleId"></
       xs:element>
    <xs:element name="Name" type="ActorRoleName"></xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="CompositeActorRoleGuid" use="required"/>
</xs:complexType>
```

Listing D.30: Composite actor role

```
<xs:complexType name="TransactionProcessStepKind">
  <xs:annotation>
    <xs:documentation>
      The Independent Fact Kind corresponds to the Transaction Kind in a one-
         on-one relation.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="TransactionKind" type="TransactionKindGuid"/>
    <xs:element name="StepKind" type="GeneralStepKind"/>
```

```
  </xs:sequence>
  <xs:attribute name="Id" type="TransactionProcessStepKindGuid" use="required
      "/>
</xs:complexType>
```

Listing D.31: Transaction process step kind

```
<xs:complexType name="IndependentFactKind">
  <xs:annotation>
    <xs:documentation>
      The Independent Fact Kind corresponds to the Transaction Kind in a one-
          on-one relation.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Identification" type="ProductKindId"/>
    <xs:element name="Formulation" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="Id" type="IndependentFactKindGuid" use="required"/>
</xs:complexType>
```

Listing D.32: Independent fact kind

```
<xs:complexType name="EntityType">
  <xs:sequence>
    <!--<xs:element name="Identification" type="entity_type_id"></xs:element>
        -->
    <xs:element name="Name" type="DefiniteEntityVariable"></xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="EntityTypeGuid" use="required"/>
</xs:complexType>
```

Listing D.33: Entity type

Listing D.34: Property type

```
<xs:complexType name="AttributeType">
  <xs:sequence>
    <!--<xs:element name="Identification" type="entity_type_id"></xs:element>
        -->
    <xs:element name="Name" type="FactKindName"></xs:element>
    <xs:element name="ValueTypes" type="ObjectClassName"></xs:element>
    <xs:element name="EntityType" type="EntityTypeGuid"/>
  </xs:sequence>
  <xs:attribute name="Id" type="AttributeTypeGuid" use="required"/>
</xs:complexType>
```

Listing D.35: Attribute type

D.8 Element Guid's

All guids are based on the Guid datatype

```
<xs:simpleType name="Guid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a GUID, generally the id of an element.
      Sometimes, notations include curly brackets. In this standard we use
          GUIDs without those brackets.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0
        -9]{4}-[a-fA-F0-9]{12}"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.36: Guid type

```
<xs:simpleType name="TransactionKindGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a TransactionKindId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.37: Transaction kind guid

```
<xs:simpleType name="TransactionProcessStepKindGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a TransactionProcessStepKindId, generally the id
          of an element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.38: Transaction step kind guid

```
<xs:simpleType name="IndependentFactKindGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a IndependentFactKindId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.39: Independent fact kind guid

```
<xs:simpleType name="AggregateTransactionKindGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a AggregateTransactionKindId, generally the id of
           an element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.40: Aggregated transaction kind guid

```
<xs:simpleType name="ElementaryActorRoleGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ElementaryActorRoleId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.41: Elementary actor role guid

```
<xs:simpleType name="CompositeActorRoleGuid">
  <xs:annotation>
```

```
    <xs:documentation xml:lang="en">
      The representation of a CompositeActorRoleId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.42: Composite actor role guid

```
<xs:simpleType name="EntityTypeGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a EntityTypeId, generally the id of an element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.43: Entity type guid

```
<xs:simpleType name="ActorGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ActorId, generally the id of an element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.44: Actor guid

```
<xs:simpleType name="AttributeTypeGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a AttributeTypeId, generally the id of an element
          .
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
```

```
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.45: Attribute type guid

```
<xs:simpleType name="ActionRuleTypeGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ActionRuleTypeId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.46: Action rule type guid

```
<xs:simpleType name="ConnectionGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ConnectionId, generally the id of an element.
      This type is used to distinguish between usage of guids between element
           types and connections. No specific range is used for connection
          types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.47: Connection guid

```
<xs:simpleType name="DiagramElementGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a DiagramElementId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between element
           types. No specific range is used for element types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
```

```
  </xs:restriction>
</xs:simpleType>
```

Listing D.48: Diagram element guid

D.9 Diagram Guid's

```
<xs:simpleType name="OrganisationConstructionDiagramGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a OrganisationConstructionDiagramId, generally
          the id of an element.
      This type is used to distinguish between usage of guids between diagram
           types. No specific range is used for diagram types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.49: Organisation construction diagram guid

```
<xs:simpleType name="ProcessStructureDiagramGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ProcessStructureDiagramGuid, generally the id
          of an element.
      This type is used to distinguish between usage of guids between diagram
           types. No specific range is used for diagram types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.50: Process structure diagram guid

```
<xs:simpleType name="ActorFunctionDiagramGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ActionFunctionDiagramId, generally the id of an
           element.
      This type is used to distinguish between usage of guids between diagram
           types. No specific range is used for diagram types.
    </xs:documentation>
  </xs:annotation>
```

```
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.51: Actor function diagram guid

```
<xs:simpleType name="ObjectFactDiagramGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ObjectFactDiagramId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between diagram
           types. No specific range is used for diagram types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.52: Object fact diagram guid

```
<xs:simpleType name="TransactionProcessDiagramGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a TransactionProcessDiagramId, generally the id
          of an element.
      This type is used to distinguish between usage of guids between diagram
           types. No specific range is used for diagram types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.53: Transaction process diagram guid

```
<xs:simpleType name="ActionRuleDiagramGuid">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The representation of a ActionRuleDiagramId, generally the id of an
          element.
      This type is used to distinguish between usage of guids between diagram
           types. No specific range is used for diagram types.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="Guid">
  </xs:restriction>
</xs:simpleType>
```

Listing D.54: Action rule diagram guid

D.10 Element identification

```
<xs:simpleType name="TransactionKindId">
  <xs:annotation>
    <xs:documentation>
      The transaction_kind_id starts with a capital T followed by an positive
           integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
      For DEMO4 transactions can start with a TK.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="T[K]?[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.55: Transaction kind id

```
<xs:simpleType name="AggregateTransactionKindId">
  <xs:annotation>
    <xs:documentation>
      The aggregate_transaction_kind_id starts with a capital AT followed by
          an positive integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="AT[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.56: Aggregated transaction kind id

```
<xs:simpleType name="ActorRoleId">
  <xs:annotation>
    <xs:documentation>
      The actor_role_id starts with a capital A followed by an positive
          integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
```

```
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="A[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.57: Actor role id

```
<xs:simpleType name="ActorId">
  <xs:annotation>
    <xs:documentation>
      The actor_id starts with a capital ACT followed by an positive integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="ACT[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.58: Actor id

```
<xs:simpleType name="CompositeActorRoleId">
  <xs:annotation>
    <xs:documentation>
      The composite_actor_role_id starts with a capital CA followed by an
          positive integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="CA[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.59: Composite actor role id

```
<xs:simpleType name="FactKindId">
  <xs:annotation>
    <xs:documentation>
      The fact_kind_id starts with a capital F followed by an positive
          integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
    </xs:documentation>
```

```
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="F[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.60: Fact kind id

```
<xs:simpleType name="ProductKindId">
  <xs:annotation>
    <xs:documentation>
      The ProductKindId starts with a capital P followed by an positive
          integer.
      Optionally followed by a dash and another positive integer.
      The positive integers can have leading zeros.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="P[0-9]+([-][0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.61: Product kind id

D.11 Enumerations

```
<xs:simpleType name="TransactionSort">
  <xs:annotation>
    <xs:documentation>
      The transactionsort is enumerated to list all possible names.
      The enumeration "physical" is derived from the PhD of Joop De Jong.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="original"/>
    <xs:enumeration value="informational"/>
    <xs:enumeration value="documental"/>
    <xs:enumeration value="physical"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.62: Transaction sorts

```
<xs:simpleType name="Cardinality">
  <xs:annotation>
```

```
    <xs:documentation>
      The cardinality options for a connection can be any number or infinity.
      Infinity is represented with a "*".
      Cardinality is given on both ends of the connection.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[1-9][0-9]*|[*]"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.63: Cardinality

```
<xs:simpleType name="GeneralStepKind">
  <xs:annotation>
    <xs:documentation>
      The general step kind is enumerated to list all possible transaction
          step abbreviations.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Initial">
      <xs:annotation>
        <xs:documentation>
          The initial step. This is where the process within a transaction
              starts and ends.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="Request">
      <xs:annotation>
        <xs:documentation>
          The request step.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="Requested">
      <xs:annotation>
        <xs:documentation>
          The requested state.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="Promise">
      <xs:annotation>
        <xs:documentation>
          The promise step.
```

```
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Promised">
  <xs:annotation>
    <xs:documentation>
      The promised state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Execute">
  <xs:annotation>
    <xs:documentation>
      The execute step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Executed">
  <xs:annotation>
    <xs:documentation>
      The execute state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="State">
  <xs:annotation>
    <xs:documentation>
      The state step. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Declare">
  <xs:annotation>
    <xs:documentation>
      The state step in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Stated">
  <xs:annotation>
    <xs:documentation>
      The stated state. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Declared">
  <xs:annotation>
```

```
    <xs:documentation>
      The stated state in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Accept">
  <xs:annotation>
    <xs:documentation>
      The accept step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Accepted">
  <xs:annotation>
    <xs:documentation>
      The accepted state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Decline">
  <xs:annotation>
    <xs:documentation>
      The decline step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Declined">
  <xs:annotation>
    <xs:documentation>
      The declined state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Quit">
  <xs:annotation>
    <xs:documentation>
      The quit step. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Quited">
  <xs:annotation>
    <xs:documentation>
      The quited state. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
```

```
<xs:enumeration value="Reject">
  <xs:annotation>
    <xs:documentation>
      The reject step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Rejected">
  <xs:annotation>
    <xs:documentation>
      The reject state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Stop">
  <xs:annotation>
    <xs:documentation>
      The stop step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="Stopped">
  <xs:annotation>
    <xs:documentation>
      The stopped state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>

<xs:enumeration value="RevokeRequest">
  <xs:annotation>
    <xs:documentation>
      The revoke request step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokedRequest">
  <xs:annotation>
    <xs:documentation>
      The revoke requested state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeRequestAllow">
  <xs:annotation>
    <xs:documentation>
      The revoke request allow step.
```

```
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeRequestAllowed">
  <xs:annotation>
    <xs:documentation>
      The revoke request allowed state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeRequestRefuse">
  <xs:annotation>
    <xs:documentation>
      The revoke request refuse step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeRequestRefused">
  <xs:annotation>
    <xs:documentation>
      The revoke request refused state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>

<xs:enumeration value="RevokePromise">
  <xs:annotation>
    <xs:documentation>
      The revoke promise step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokedPromise">
  <xs:annotation>
    <xs:documentation>
      The revoke promised state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokePromiseAllow">
  <xs:annotation>
    <xs:documentation>
      The revoke promise allow step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokePromiseAllowed">
```

```
  <xs:annotation>
    <xs:documentation>
      The revoke promise allowed state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokePromiseRefuse">
  <xs:annotation>
    <xs:documentation>
      The revoke promise refuse step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokePromiseRefused">
  <xs:annotation>
    <xs:documentation>
      The revoke promise refused state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>

<xs:enumeration value="RevokeState">
  <xs:annotation>
    <xs:documentation>
      The revoke state step. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokedState">
  <xs:annotation>
    <xs:documentation>
      The revoke stated state. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeStateAllow">
  <xs:annotation>
    <xs:documentation>
      The revoke state allow step. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeStateAllowed">
  <xs:annotation>
    <xs:documentation>
      The revoke state allowed state. Does not exist in DEMO 4.
    </xs:documentation>
```

```
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeStateRefuse">
  <xs:annotation>
    <xs:documentation>
      The revoke state refuse step. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeStateRefused">
  <xs:annotation>
    <xs:documentation>
      The revoke state refused state. Does not exist in DEMO 4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>

<xs:enumeration value="RevokeDeclare">
  <xs:annotation>
    <xs:documentation>
      The revoke state step in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokedDeclare">
  <xs:annotation>
    <xs:documentation>
      The revoke stated state in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeDeclareAllow">
  <xs:annotation>
    <xs:documentation>
      The revoke state allow step in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeDeclareAllowed">
  <xs:annotation>
    <xs:documentation>
      The revoke state allowed state in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeDeclareRefuse">
  <xs:annotation>
```

```
    <xs:documentation>
      The revoke state refuse step in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeDeclareRefused">
  <xs:annotation>
    <xs:documentation>
      The revoke state refused state in DEMO4.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>

<xs:enumeration value="RevokeAccept">
  <xs:annotation>
    <xs:documentation>
      The revoke accept step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokedAccept">
  <xs:annotation>
    <xs:documentation>
      The revoke accepted state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeAcceptAllow">
  <xs:annotation>
    <xs:documentation>
      The revoke accept allow step.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeAcceptAllowed">
  <xs:annotation>
    <xs:documentation>
      The revoke accept allowed state.
    </xs:documentation>
  </xs:annotation>
</xs:enumeration>
<xs:enumeration value="RevokeAcceptRefuse">
  <xs:annotation>
    <xs:documentation>
      The revoke accept refuse step.
    </xs:documentation>
  </xs:annotation>
```

```
    </xs:enumeration>
    <xs:enumeration value="RevokeAcceptRefused">
      <xs:annotation>
        <xs:documentation>
          The revoke accept refused state.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>
```

Listing D.64: General Step Kind

D.12 Names

```
<xs:simpleType name="TransactionKindName">
  <xs:annotation>
    <xs:documentation>
      The transaction_kind_name are lower case words separated by spaces.
      The name can be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]*([␣][a-z]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.65: Transaction kind name

```
<xs:simpleType name="ActorRoleName">
  <xs:annotation>
    <xs:documentation>
      The actor_role_name are lower case words separated by spaces.
      The name can be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]*([␣][a-z]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.66: Actor role name

```
<xs:simpleType name="ActorName">
  <xs:annotation>
    <xs:documentation>
```

```
      The actor_role_name are lower case words separated by spaces.
      The name can be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]*([␣][a-z]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.67: Actor name

```
<xs:simpleType name="ObjectClassName">
  <xs:annotation>
    <xs:documentation>
      The object_class_name are upper case words separated by spaces.
      The name cannot be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z]+([␣][A-Z]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.68: Object class name

```
<xs:simpleType name="FactKindName">
  <xs:annotation>
    <xs:documentation>
      The fact_kind_name are lower case words separated by spaces.
      The name can be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]+([␣][a-z]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.69: Fact kind name

```
<xs:simpleType name="DefiniteEntityVariable">
  <xs:annotation>
    <xs:documentation>
      The definite_entity_variable are "lower␣case␣words␣between␣brackets" or
          "capitalised␣words".
      [entity] | ENTITY
      The name can not be empty.
    </xs:documentation>
  </xs:annotation>
```

```
  <xs:restriction base="xs:string">
    <xs:pattern value="\[[a-z]+([␣][a-z]+)*\]|[A-Z][a-z]*([␣][A-Z][a-z]*)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.70: Definite Entity Variable name

```
<xs:simpleType name="ActionRuleName">
  <xs:annotation>
    <xs:documentation>
      The ActionRuleName are "lower␣case␣words␣between␣pointy␣hooks".
      &lt;condition&gt;
      The name can not be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[&lt;][a-z]+([␣][a-z]+)*[&gt;]"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.71: Action rule name

```
<xs:simpleType name="DEMOModelName">
  <xs:annotation>
    <xs:documentation>
      The DEMOModelName are "lower␣case␣words␣separated␣by␣spaces".
      The name can not be empty.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z]+([␣][a-zA-Z]+)*"/>
  </xs:restriction>
</xs:simpleType>
```

Listing D.72: DEMO model name

D.13 Relation Elements

```
<xs:complexType name="Connection">
  <xs:sequence>
    <xs:choice>
```

Listing D.73: Contained in EAR

```
      <xs:element name="EARContainedInCAR">
        <xs:complexType>
```

```
    <xs:sequence>
      <xs:element name="FromElementaryActorRole" type="
          ElementaryActorRoleGuid">
      </xs:element>
      <xs:element name="ToCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.74: EAR contained in CAR

```
<xs:element name="CARContainedInCAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
      <xs:element name="ToCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.75: CAR contained in CAR

```
<xs:element name="ETKContainedInCAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionKind" type="TransactionKindGuid">
      </xs:element>
      <xs:element name="ToCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.76: ETK contained in CAR

```
<xs:element name="ETKContainedInATK">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionKind" type="TransactionKindGuid">
      </xs:element>
      <xs:element name="ToAggregateTransactionKind" type="
          AggregateTransactionKindGuid">
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.77: ETK contained in ATK

```
<xs:element name="ConcernsETTK">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromEntityType" type="EntityTypeGuid">
      </xs:element>
      <xs:element name="ToTransactionKind" type="TransactionKindGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.78: Concerns

```
<xs:element name="InitiatorEAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromElementaryActorRole" type="
          ElementaryActorRoleGuid">
      </xs:element>
      <xs:element name="ToTransactionKind" type="TransactionKindGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.79: Initiator EAR

```
<xs:element name="InitiatorCAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
      <xs:element name="ToTransactionKind" type="TransactionKindGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.80: Initiator CAR

```
<xs:element name="ExecutorEAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionKind" type="TransactionKindGuid">
      </xs:element>
      <xs:element name="ToElementaryActorRole" type="
          ElementaryActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.81: Executor EAR

```
<xs:element name="ExecutorCAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionKind" type="TransactionKindGuid">
      </xs:element>
      <xs:element name="ToCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.82: Executor CAR

```
<xs:element name="InterstrictionATEAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromAggregateTransactionKind" type="
          AggregateTransactionKindGuid">
      </xs:element>
      <xs:element name="ToElementaryActorRole" type="
          ElementaryActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.83: Access AT to EAR

```
<xs:element name="InterstrictionATCAR" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromAggregateTransactionKind" type="
          AggregateTransactionKindGuid">
```

```
      </xs:element>
      <xs:element name="ToCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.84: Access AT to CAR

```
<xs:element name="InterstrictionTEAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionKind" type="TransactionKindGuid">
      </xs:element>
      <xs:element name="ToElementaryActorRole" type="
          ElementaryActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.85: Access TK to EAR

```
<xs:element name="InterstrictionTCAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionKind" type="TransactionKindGuid">
      </xs:element>
      <xs:element name="ToCompositeActorRole" type="
          CompositeActorRoleGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.86: Access TK to CAR

```
<xs:element name="SpecializationET">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromEntityType" type="EntityTypeGuid">
      </xs:element>
      <xs:element name="ToEntityType" type="EntityTypeGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.87: Specialisation ET

```
<xs:element name="AggregationET">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromEntityType" type="EntityTypeGuid">
      </xs:element>
      <xs:element name="ToEntityType" type="EntityTypeGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.88: Aggregation ET

```
<xs:element name="GeneralizationET">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromEntityType" type="EntityTypeGuid">
      </xs:element>
      <xs:element name="ToEntityType" type="EntityTypeGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.89: Generalisation of ET

```
<xs:element name="ExcludesET">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromEntityType" type="EntityTypeGuid">
      </xs:element>
      <xs:element name="ToEntityType" type="EntityTypeGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.90: Excludes ET

```
<xs:element name="ExcludesRF">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromConnection" type="ConnectionGuid">
      </xs:element>
      <xs:element name="ToConnection" type="ConnectionGuid">
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.91: Excludes relation

```
<xs:element name="WaitConditionTPSK">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionProcessStepKind" type="
          TransactionProcessStepKindGuid">
      </xs:element>
      <xs:element name="ToTransactionProcessStepKind" type="
          TransactionProcessStepKindGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.92: Wait TPSK

```
<xs:element name="InitiationTPSK">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromTransactionProcessStepKind" type="
          TransactionProcessStepKindGuid">
      </xs:element>
      <xs:element name="ToTransactionProcessStepKind" type="
          TransactionProcessStepKindGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.93: Initiation TPSK

```
<xs:element name="RoleOfEAR">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FromElementaryActorRole" type="
          ElementaryActorRoleGuid">
      </xs:element>
      <xs:element name="ToActor" type="ActorGuid">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing D.94: Role of EAR

```
      <xs:element name="RoleOfCAR">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="FromCompositeActorRole" type="
                CompositeActorRoleGuid">
            </xs:element>
            <xs:element name="ToActor" type="ActorGuid">
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

Listing D.95: Role of CAR

```
<xs:complexType name="Connection">
  <xs:sequence>
    <xs:choice>
```

Listing D.96: Connections (start)

```
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="Id" type="ConnectionGuid" use="required"/>
  <xs:attribute name="Name" type="FactKindName" use="optional"/>
  <xs:attribute name="FromCardinality" type="Cardinality" use="required"/>
  <xs:attribute name="ToCardinality" type="Cardinality" use="required"/>
</xs:complexType>
```

Listing D.97: Connections (end)

D.14 Diagram Elements

```
<xs:complexType name="TransactionKindElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="TransactionKindGuid"/>
    <xs:element name="Initiator" type="xs:boolean"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.98: Transaction kind element

```
<xs:complexType name="IndependentFactKindElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="IndependentFactKindGuid"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.99: Independant fact element

```
<xs:complexType name="TransactionProcessStepKindElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="TransactionProcessStepKindGuid"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.100: Transaction proces step kind element

```
<xs:complexType name="AggregateTransactionKindElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="AggregateTransactionKindGuid"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.101: Aggregate transaction kind element

```
<xs:complexType name="ElementaryActorRoleElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="ElementaryActorRoleGuid"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.102: Elementary actor role element

```
<xs:complexType name="CompositeActorRoleElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="CompositeActorRoleGuid"/>
    <xs:element name="Location" type="Location"/>
```

```
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.103: Composite actor role element

```
<xs:complexType name="ActorElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="CompositeActorRoleGuid"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.104: Actor element

```
<xs:complexType name="EntityTypeElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="EntityTypeGuid"/>
    <xs:element name="Location" type="Location"/>
    <xs:element name="Size" type="Size"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.105: Entity type element

```
<xs:complexType name="ConnectionElement">
  <xs:sequence>
    <xs:element name="ReferenceId" type="ConnectionGuid"/>
    <xs:element name="Hidden" type="xs:boolean"/>
    <xs:element name="Line" type="Line"/>
  </xs:sequence>
  <xs:attribute name="Id" type="DiagramElementGuid" use="required"/>
</xs:complexType>
```

Listing D.106: Connection element

D.15 Validation Result

```
<xs:complexType name="ValidationResult">
  <xs:all>
    <xs:element name="Number" type="xs:string" minOccurs="0" maxOccurs="1"></
        xs:element>
```

```
    <xs:element name="Message" type="xs:string" minOccurs="0" maxOccurs="1"><
       /xs:element>
    <xs:element name="Id" type="Guid" minOccurs="0" maxOccurs="1"></
       xs:element>
    <xs:element name="Identification" type="xs:string" minOccurs="0"
       maxOccurs="1"></xs:element>
    <xs:element name="Name" type="xs:string" minOccurs="0" maxOccurs="1"></
       xs:element>
  </xs:all>
</xs:complexType>
```

Listing D.107: Validation results

D.16 Diagrams

```
<xs:complexType name="OrganisationConstructionDiagram">
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Formulation" type="xs:string"/>
    <xs:element name="TransactionKindElements" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Transaction Kinds Elements in this
             diagram.
          All references to Transaction Kinds must be available in the
             DEMOmodel\TransactionKinds List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="TransactionKindElement" type="
             TransactionKindElement" minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                A Transaction Kind Element can be added. When the
                   TransactionKindElements is present,
                a minimum of one TransactionKindElement must be added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="AggregateTransactionKindElements" minOccurs="0"
       maxOccurs="1">
      <xs:annotation>
```

```
    <xs:documentation>
      A listing of all available Aggregate Transaction Kinds Elements.
      All references to Aggregate Transaction Kinds must be available in
          the DEMOmodel\AggregateTransactionKinds List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="AggregateTransactionKindElement" type="
          AggregateTransactionKindElement" minOccurs="1" maxOccurs="
          unbounded">
        <xs:annotation>
          <xs:documentation>
            A Aggregate Transaction Kind Element can be added. When the
                AggregateTransactionKindElements is present,
            a minimum of one AggregateTransactionKindElement must be added
                .
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ElementaryActorRoleElements" minOccurs="0" maxOccurs="1
    ">
  <xs:annotation>
    <xs:documentation>
      A listing of all available Elementary Actor Role Elements.
      All references to Elementary Actor Roles must be available in the
          DEMOmodel\ElementaryActorRoles List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ElementaryActorRoleElement" type="
          ElementaryActorRoleElement" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Elementary Actor Role Element can be added. When the
                ElementaryActorRoleElements is present,
            a minimum of one ElementaryActorRoleElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="CompositeActorRoleElements" minOccurs="0" maxOccurs="1"
    >
  <xs:annotation>
    <xs:documentation>
      A listing of all available Composite Actor Role Elements.
      All references to Composite Actor Roles must be available in the
          DEMOmodel\CompositeActorRoles List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CompositeActorRoleElement" type="
          CompositeActorRoleElement" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Composite Actor Role Element can be added. When the
                CompositeActorRoleElements is present,
            a minimum of one CompositeActorRoleElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ConnectionElements" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      A listing of all available Connection Elements.
      All references to Connections must be available in the DEMOmodel\
          Connections List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ConnectionElement" type="ConnectionElement"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Connection Element can be added. When the ConnectionElements
                is present,
            a minimum of one ConnectionElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
  </xs:all>
  <xs:attribute name="Id" type="OrganisationConstructionDiagramGuid" use="
     required"/>
</xs:complexType>
```

Listing D.108: Construction diagram

```
<xs:complexType name="ProcessStructureDiagram">
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Formulation" type="xs:string"/>
    <xs:element name="TransactionKindElements" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Transaction Kind Elements.
          All references to TransactionKinds must be available in the
              DEMOmodel\TransactionKinds List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="TransactionKindElement" type="
              TransactionKindElement" minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                A Transaction Kind Elements can be added. When the
                    TransactionKindElements is present,
                a minimum of one TransactionKindElement must be added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ConnectionElements" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Connection Elements.
          All references to Connections must be available in the DEMOmodel\
              Connections List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ConnectionElement" type="ConnectionElement"
              minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
```

```
            <xs:documentation>
              A Connection Element can be added. When the ConnectionElements
                  is present,
              a minimum of one ConnectionElement must be added.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:all>
 <xs:attribute name="Id" type="ProcessStructureDiagramGuid" use="required"/>
</xs:complexType>
```

Listing D.109: Process structure diagram

```
<xs:complexType name="TransactionProcessDiagram">
 <xs:sequence>
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Formulation" type="xs:string"/>
  <xs:element name="TransactionProcessStepKindElements" minOccurs="0"
      maxOccurs="1">
   <xs:annotation>
    <xs:documentation>
     A listing of all available Transaction Process Step Kind Elements.
     All references to TransactionProcessStepKinds must be available in
         the DEMOmodel\TransactionProcessStepKinds List.
    </xs:documentation>
   </xs:annotation>
   <xs:complexType>
    <xs:sequence>
     <xs:element name="TransactionProcessStepKindElement" type="
         TransactionProcessStepKindElement" minOccurs="1" maxOccurs="
         unbounded">
      <xs:annotation>
       <xs:documentation>
        A Transaction Process Step Kind Element can be added. When the
            TransactionProcessStepKindElements is present,
        a minimum of one TransactionProcessStepKindElement must be
            added.
       </xs:documentation>
      </xs:annotation>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="ConnectionElements" minOccurs="0" maxOccurs="1">
```

```
      <xs:annotation>
        <xs:documentation>
          A listing of all available Connection Elements.
          All references to Connections must be available in the DEMOmodel\
              Connections List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ConnectionElement" type="ConnectionElement"
              minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                A Connection Element can be added. When the ConnectionElements
                    is present,
                a minimum of one ConnectionElement must be added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="TransactionProcessDiagramGuid" use="required"
      />
</xs:complexType>
```

Listing D.110: Transaction Process diagram

```
<xs:complexType name="ObjectFactDiagram">
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Formulation" type="xs:string"/>
    <xs:element name="EntityTypeElements" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Entity Type Elements.
          All references to EntityTypes must be available in the DEMOmodel\
              EntityTypes List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="EntityTypeElement" type="EntityTypeElement"
              minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
```

```
            A Entity Type Elements can be added. When the
                EntityTypeElements is present,
            a minimum of one EntityTypeElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ConnectionElements" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      A listing of all available Connection Elements.
      All references to Connections must be available in the DEMOmodel\
          Connections List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ConnectionElement" type="ConnectionElement"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Connection Element can be added. When the ConnectionElements
                is present,
            a minimum of one ConnectionElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:all>
<xs:attribute name="Id" type="ObjectFactDiagramGuid" use="required"/>
</xs:complexType>
```

Listing D.111: Process structure diagram

```
<xs:complexType name="ActionRuleDiagram">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Formulation" type="xs:string"/>
    <xs:element name="TransactionProcessStepKindElements" minOccurs="0"
        maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Transaction Process Step Kind Elements.
```

```
        All references to TransactionProcessStepKinds must be available in
            the DEMOmodel\TransactionProcessStepKinds List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="TransactionProcessStepKindElement" type="
              TransactionProcessStepKindElement" minOccurs="1" maxOccurs="
              unbounded">
            <xs:annotation>
              <xs:documentation>
                A Transaction Process Step Kind Element can be added. When the
                     TransactionProcessStepKindElements is present,
                a minimum of one TransactionProcessStepKindElement must be
                    added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ConnectionElements" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Connection Elements.
          All references to Connections must be available in the DEMOmodel\
              Connections List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ConnectionElement" type="ConnectionElement"
              minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                A Connection Element can be added. When the ConnectionElements
                     is present,
                a minimum of one ConnectionElement must be added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="ActionRuleDiagramGuid" use="required"/>
</xs:complexType>
```

Listing D.112: Action Rule Diagram

```
<xs:complexType name="ActorFunctionDiagram">
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Formulation" type="xs:string"/>
    <xs:element name="ActorElements" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Actor Elements.
          All references to Actor must be available in the DEMOmodel\Actor
              List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ActorElement" type="ActorElement" minOccurs="1"
              maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                A Actor Elements can be added. When the ActorElements is
                    present,
                a minimum of one ActorElements must be added.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ElementaryActorRoleElements" minOccurs="0" maxOccurs="1
        ">
      <xs:annotation>
        <xs:documentation>
          A listing of all available Elementary Actor Role Elements.
          All references to Elementary Actor Roles must be available in the
              DEMOmodel\ElementaryActorRoles List.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ElementaryActorRoleElement" type="
              ElementaryActorRoleElement" minOccurs="1" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                A Elementary Actor Role Element can be added. When the
                    ElementaryActorRoleElements is present,
```

```
            a minimum of one ElementaryActorRoleElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="CompositeActorRoleElements" minOccurs="0" maxOccurs="1"
    >
  <xs:annotation>
    <xs:documentation>
      A listing of all available Composite Actor Role Elements.
      All references to Composite Actor Roles must be available in the
          DEMOmodel\CompositeActorRoles List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CompositeActorRoleElement" type="
          CompositeActorRoleElement" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Composite Actor Role Element can be added. When the
                CompositeActorRoleElements is present,
            a minimum of one CompositeActorRoleElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ConnectionElements" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      A listing of all available Connection Elements.
      All references to Connections must be available in the DEMOmodel\
          Connections List.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ConnectionElement" type="ConnectionElement"
          minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            A Connection Element can be added. When the ConnectionElements
                is present,
```

```
            a minimum of one ConnectionElement must be added.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:all>
<xs:attribute name="Id" type="ActorFunctionDiagramGuid" use="required"/>
</xs:complexType>
```

Listing D.113: Actor function diagram

D.17 Action Rules

```
<xs:complexType name="ActionRuleType">
  <xs:sequence>
    <xs:element name="Name" type="ActionRuleName"></xs:element>
    <xs:element name="Event" type="EventPartType" minOccurs="1" maxOccurs="1"
        >
    </xs:element>
    <xs:element name="Assess" type="AssessPartType" minOccurs="0" maxOccurs="
        1">
    </xs:element>
    <xs:element name="Response" type="IfType" minOccurs="1" maxOccurs="1">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="ActionRuleTypeGuid" use="required"/>
</xs:complexType>
```

Listing D.114: Action Rule Type

```
<xs:complexType name="EventPartType">
  <xs:sequence>
    <xs:element name="Agendum" type="ActionType" minOccurs="1" maxOccurs="1">
    </xs:element>
    <xs:element name="While" type="BlockType" minOccurs="0" maxOccurs="1">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.115: Event Part Type

```
<xs:complexType name="AssessPartType">
  <xs:sequence>
    <xs:element name="Justice" type="BlockType" minOccurs="0" maxOccurs="
        unbounded">
```

```
    </xs:element>
    <xs:element name="Sincerity" type="BlockType" minOccurs="0" maxOccurs="
        unbounded">
    </xs:element>
    <xs:element name="Truth" type="FlowType" minOccurs="0" maxOccurs="1">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.116: Assess Part Type

```
<xs:complexType name="ActionType">
  <xs:sequence>
    <xs:element name="TransactionProcessStepKind" type="
        TransactionProcessStepKindGuid" minOccurs="0" maxOccurs="1">
    </xs:element>
    <xs:element name="With" type="BlockType" minOccurs="0" maxOccurs="
        unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.117: Action Type

```
<xs:complexType name="FlowType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="Foreach" type="ForeachType">
      </xs:element>
      <xs:element name="If" type="IfType">
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Listing D.118: Flow Type

```
<xs:complexType name="IfType">
  <xs:sequence>
    <xs:element name="Condition" type="ConditionType" minOccurs="1" maxOccurs
        ="1">
    </xs:element>
    <xs:element name="Then" type="BlockType" minOccurs="1" maxOccurs="
        unbounded">
    </xs:element>
    <xs:element name="Else" type="BlockType" minOccurs="0" maxOccurs="
        unbounded">
    </xs:element>
```

```
  </xs:sequence>
</xs:complexType>
```

Listing D.119: If Type

```
<xs:complexType name="BlockType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="Flow" type="FlowType" minOccurs="0" maxOccurs="1">
      </xs:element>
      <xs:element name="Action" type="ActionType" minOccurs="0" maxOccurs="1"
          >
      </xs:element>
      <xs:element name="AttributeProperty" type="FmReferenceType" minOccurs="
          0" maxOccurs="1">
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Listing D.120: Block Type

```
<xs:complexType name="ForeachType">
  <xs:sequence>
    <xs:element name="Element" type="FactKindName" minOccurs="1" maxOccurs="1
        ">
    </xs:element>
    <xs:element name="Collection" type="EntityTypeGuid" minOccurs="1"
        maxOccurs="1">
    </xs:element>
    <xs:element name="Repeating" type="BlockType" minOccurs="1" maxOccurs="
        unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.121: For Each Type

```
<xs:complexType name="ConditionType">
  <xs:choice>
    <xs:sequence>
      <xs:element name="Left" type="ConditionType" minOccurs="1" maxOccurs="1
          ">
      </xs:element>
      <xs:element name="Compare" type="xs:string" minOccurs="1" maxOccurs="1"
          >
      </xs:element>
      <xs:element name="Right" type="ConditionType" minOccurs="1" maxOccurs="
          1">
```

```
      </xs:element>
    </xs:sequence>
    <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1">
    </xs:element>
    <xs:element name="FmReference" type="FmReferenceType" minOccurs="1"
       maxOccurs="1">
    </xs:element>
  </xs:choice>
</xs:complexType>
```

Listing D.122: Condition Type

```
<xs:complexType name="FmReferenceType">
  <xs:sequence>
    <xs:element name="Entity" type="EntityTypeGuid" minOccurs="1" maxOccurs="
       1">
    </xs:element>
    <xs:element name="Attribute" type="AttributeTypeGuid" minOccurs="0"
       maxOccurs="1">
    </xs:element>
    <xs:element name="Connection" type="ConnectionGuid" minOccurs="0"
       maxOccurs="1">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Listing D.123: Fact Model Reference Type

D.18 Extensions

```
      <xs:element name="ExtendedInfo" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:any minOccurs="0" maxOccurs="unbounded" processContents="lax"
               namespace="##any" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

Listing D.124: Extended info collection

Appendix E

Logical Metamodel

E.1 Core

$$\begin{aligned} \mathsf{DEMO}(x) \implies & \mathsf{ConstructionModel}(x) \\ \vee\ & \mathsf{ProcessModel}(x) \\ \vee\ & \mathsf{FactModel}(x) \\ \vee\ & \mathsf{ActionModel}(x) \end{aligned} \tag{E.1}$$ DEMO

E.2 Construction Metamodel

$$\mathsf{ConstructionModel}(x) \implies \mathsf{ConstructionConcept}(x) \vee \mathsf{ConstructionRelation}(x) \tag{E.2}$$ CM

$$\mathsf{ConstructionConcept}(x) \implies \mathsf{TransactionKind}(x) \vee \mathsf{ActorRole}(x) \tag{E.3}$$ CMcs

$$\begin{aligned}\mathsf{ConstructionRelation}(\langle s,t\rangle) \implies & \mathsf{executor}(s,t) \\ \vee\ & \mathsf{initiator}(s,t) \\ \vee\ & \mathsf{access}(s,t) \\ \vee\ & \mathsf{TKcontainedinATK}(s,t) \\ \vee\ & \mathsf{TKcontainedinCAR}(s,t) \\ \vee\ & \mathsf{ARaccessATK}(s,t) \\ \vee\ & \mathsf{EARcontainedinCAR}(s,t) \\ \vee\ & \mathsf{CARcontainedinCAR}(s,t)\end{aligned} \tag{E.4}$$ CMrel

$$\mathsf{ActorRole}(x) \implies \mathsf{ElementaryActorRole}(x) \veebar \mathsf{CompositeActorRole}(x) \tag{E.5}$$ AR

$$\mathsf{TransactionKind}(x) \implies \mathsf{ElementaryTransactionKind}(x) \veebar \mathsf{AggregateTransactionKind}(x) \tag{E.6}$$ TK

ETK

$$\mathsf{executor}(x,y) \implies \mathsf{TransactionKind}(x) \wedge \mathsf{ActorRole}(y) \tag{E.7}$$ TKe

$$\mathsf{initiator}(x,y) \implies \mathsf{ActorRole}(x) \wedge \mathsf{TransactionKind}(y) \tag{E.8}$$ TKi

$$\mathsf{TKcontainedinATK}(x, y) \implies \mathsf{ElementaryTransactionKind}(x) \wedge \mathsf{AggregateTransactionKind}(y) \quad \text{(E.9)}\ \text{TKinATK}$$

$$\mathsf{TKcontainedinCAR}(x, y) \implies \mathsf{ElementaryTransactionKind}(x) \wedge \mathsf{CompositeActorRole}(y) \quad \text{(E.10)}\ \text{TKinCAR}$$

$$\mathsf{TKcontainedinCAR}(x, y) \wedge \mathsf{initiator}(p, x) \wedge \mathsf{ElementaryActorRole}(p) \implies \mathsf{EARcontainedinCAR}(p, y) \quad \text{(E.11)}\ \text{TKinCARi}$$

$$\mathsf{TKcontainedinCAR}(x, y) \wedge \mathsf{executor}(x, p) \wedge \mathsf{ElementaryActorRole}(p) \implies \mathsf{EARcontainedinCAR}(p, y) \quad \text{(E.12)}\ \text{TKinCARe}$$

$$\mathsf{ElementaryTransactionKind}(t) \implies \exists_a \left[\mathsf{executor}(t, a)\right] \quad \text{(E.13)}\ \text{TKexistAR}$$

$$\mathsf{ElementaryTransactionKind}(t) \implies \exists_a \left[\mathsf{initiator}(t, a)\right] \quad \text{(E.14)}\ \text{TKexistARi}$$

$$\mathsf{ElementaryTransactionKind}(x) \wedge \mathsf{ElementaryActorRole}(y) \wedge \mathsf{executor}(x, y) \wedge \mathsf{initiator}(y, x) \implies \mathsf{SelfInitiator}(y) \quad \text{(E.15)}\ \text{ARself}$$

ATK

$$\mathsf{ARaccessATK}(x, y) \implies \mathsf{ActorRole}(x) \wedge \mathsf{AggregateTransactionKind}(y) \quad \text{(E.16)}\ \text{ARaccATK}$$

$$\mathsf{FTcontainedinTK}(x, y) \implies \mathsf{FactType}(x) \wedge \mathsf{TransactionKind}(y) \quad \text{(E.17)}\ \text{FTinTK}$$

EAR

$$\begin{aligned} \mathsf{EARcontainedinCAR}(x, y) \implies & \mathsf{ElementaryActorRole}(x) \\ & \wedge\ \mathsf{CompositeActorRole}(y) \end{aligned} \quad \text{(E.18)}\ \text{EARinCAR}$$

$$\begin{aligned} \mathsf{executor}(x, y) \ \wedge \\ \mathsf{executor}(x, z) \ \wedge \\ \mathsf{ElementaryActorRole}(y) \ \wedge \\ \mathsf{ElementaryActorRole}(z) \implies y = z \end{aligned} \quad \text{(E.19)}\ \text{EARexs}$$

$$\mathsf{ARaccessETK}(x, y) \implies \mathsf{ActorRole}(x) \wedge \mathsf{ElementaryTransactionKind}(y) \quad \text{(E.20)}\ \text{ARaccETK}$$

$$\begin{aligned} \mathsf{ElementaryTransactionKind}(x) \ \wedge \\ \mathsf{ElementaryActorRole}(y) \ \wedge \\ \mathsf{CompositeActorRole}(z) \ \wedge \\ \mathsf{executor}(x, y) \ \wedge \\ \mathsf{executor}(x, z) \implies \mathsf{EARcontainedinCAR}(y, z) \end{aligned} \quad \text{(E.21)}\ \text{TKme}$$

CAR

$$\begin{aligned} \mathsf{CARcontainedinCAR}(x, y) \implies & \mathsf{CompositeActorRole}(x) \\ & \wedge\ \mathsf{CompositeActorRole}(y) \end{aligned} \quad \text{(E.22)}\ \text{CARinCAR}$$

$$\nexists x, y : [\mathsf{CARcontainedinCAR}(x, y) \implies \mathsf{CARcontainedinCAR}(y, x)] \quad \text{(E.23)}\ \text{CARnocomCAR}$$

$$\mathsf{CARcontainedinCAR}(x,y) \wedge \mathsf{CARcontainedinCAR}(y,z) \implies \mathsf{CARcontainedinCAR}(x,z) \quad \text{(E.24)}\ \text{CARoverCAR}$$

$$\begin{aligned}&\mathsf{executor}(x,y) \wedge \mathsf{executor}(x,z)\\ \wedge\ &\mathsf{ElementaryTransactionKind}(x)\\ \wedge\ &\mathsf{CompositeActorRole}(y)\\ \wedge\ &\mathsf{CompositeActorRole}(z) \implies \mathsf{CARcontainedinCAR}(z,y)\end{aligned} \quad \text{(E.25)}\ \text{CARcon}$$

$$\begin{aligned}&\mathsf{EARcontainedinCAR}(x,y) \wedge\\ &\mathsf{EARcontainedinCAR}(x,z) \implies y = z\end{aligned} \quad \text{(E.26)}\ \text{EARxinCAR}$$

OCD

$$\mathsf{ocdDiagram}(d) \implies \mathsf{DiagramType}(d) \quad \text{(E.27)}\ \text{OCDd}$$

$$\mathsf{ocdDiagramElement}(x,d) \implies \mathsf{ConstructionConcept}(x) \wedge \mathsf{ocdDiagram}(d) \quad \text{(E.28)}\ \text{OCDe}$$

$$\mathsf{ocdDiagramRelation}(r,d) \implies \mathsf{ConstructionRelation}(r) \wedge \mathsf{ocdDiagram}(d) \quad \text{(E.29)}\ \text{OCDr}$$

$$\begin{aligned}\mathsf{ocdDiagramRelation}(\langle s,t\rangle) \implies\ &\mathsf{executor}(s,t)\\ \vee\ &\mathsf{initiator}(s,t)\\ \vee\ &\mathsf{access}(s,t)\\ \vee\ &\mathsf{ARaccessATK}(s,t)\end{aligned} \quad \text{(E.30)}\ \text{OCDrel}$$

TPT

$$\mathsf{tptTable}(t) \implies \mathsf{TableClass}(t) \tag{E.31}$$ TPTd

$$\mathsf{tptTableElement}(\langle x, y, z \rangle, t) \implies \mathsf{tptTable}(t) \wedge \mathsf{concern}(y, x) \wedge \mathsf{executor}(x, z) \tag{E.32}$$ TPTer

BCT

$$\mathsf{bctTable}(t) \implies \mathsf{TableClass}(t) \tag{E.33}$$ BCTd

$$\mathsf{bctTableElement}(\langle x, y \rangle, t) \implies \mathsf{bctTable}(t) \wedge \mathsf{FTcontainedinTK}(x, y) \tag{E.34}$$ BCTer

E.3 Process Metamodel

$$\mathsf{ProcessModel}(x) \implies \mathsf{ProcessConcept}(x) \lor \mathsf{ProcessRelation}(x) \tag{E.35}$$ PM

$$\mathsf{ProcessConcept}(x) \implies \mathsf{ElementaryTransactionKind}(x) \lor \mathsf{ElementaryActorRole}(x) \lor \mathsf{CompositeActorRole}(x) \lor \mathsf{TransactionProcessStepKind}(x) \tag{E.36}$$ PMcs

$$\mathsf{ProcessRelation}(\langle s, t \rangle) \implies \mathsf{TPSKpartofTK}(s, t) \lor \mathsf{TPSKinitTPSK}(s, t) \tag{E.37}$$ PMrel

PSD

$$\mathsf{psdDiagram}(d) \implies \mathsf{DiagramType}(d) \tag{E.38}$$ PSDd

$$\mathsf{psdDiagramElement}(x, d) \implies \mathsf{ProcessConcept}(x) \land \mathsf{psdDiagram}(d) \tag{E.39}$$ PSDe

$$\mathsf{psdDiagramRelation}(r, d) \implies \mathsf{ProcessRelation}(r) \land \mathsf{psdDiagram}(d) \tag{E.40}$$ PSDr

TPD

$$\mathsf{tpdDiagram}(d) \implies \mathsf{DiagramType}(d) \tag{E.41}$$ TPDd

$$\mathsf{tpdDiagramElement}(x, d) \implies \mathsf{ProcessConcept}(x) \land \mathsf{tpdDiagram}(d) \tag{E.42}$$ TPDe

$$\mathsf{tpdDiagramRelation}(r, d) \implies \mathsf{ProcessRelation}(r) \land \mathsf{tpdDiagram}(d) \tag{E.43}$$ TPDr

GSK

$$\begin{aligned}\mathsf{GeneralStepKindRequest}(r) \implies & \mathsf{GeneralStepKind}(r) \\ & \wedge\ r \in \{\mathsf{request}, \mathsf{revokerequestrequest}, \\ & \quad \mathsf{revokepromiserequest}, \mathsf{revokestaterequest}, \\ & \quad \mathsf{revokeacceptrequest}\}\end{aligned} \tag{E.44}$$ GSKr

TPSK

$$\begin{aligned}\mathsf{TransactionProcessStepKind}(x, g) \implies & \mathsf{ElementaryTransactionKind}(x) \wedge \\ & \mathsf{GeneralStepKind}(g)\end{aligned} \tag{E.45}$$ TPSK

$$\begin{aligned}& \mathsf{TransactionProcessStepKind}(x, y) \\ & \wedge\ \mathsf{TransactionProcessStepKind}(x, z) \\ & \wedge\ \mathsf{TransactionKind}(x) \\ & \wedge\ \mathsf{GeneralStepKind}(y) \\ & \wedge\ \mathsf{GeneralStepKind}(z) \\ & \implies\ y = z\end{aligned} \tag{E.46}$$ GSKse

$$\mathsf{TPSKpartofTK}(x, y) \implies \mathsf{TransactionKind}(x) \wedge \mathsf{ElementaryTransactionKind}(y) \tag{E.47}$$ TKtpsk

$$\begin{aligned}\mathsf{TPSKinitTPSK}(x, y) \implies & \exists s, t[\mathsf{TransactionProcessStepKind}(x, s) \\ & \wedge\ \mathsf{GeneralStepKind}(s) \\ & \wedge\ \mathsf{TransactionProcessStepKind}(y, t) \\ & \wedge\ \mathsf{GeneralStepKindRequest}(t)]\end{aligned} \tag{E.48}$$ TPSKtpsk

E.4 Fact Metamodel

$$\begin{aligned}\mathsf{FactModel}(x) \implies & \mathsf{FactConcept}(x) \\ \vee\ & \mathsf{FactRelation}(x)\end{aligned} \tag{E.49}$$ FM

$$\begin{aligned}\mathsf{FactConcept}(x) \implies & \mathsf{ElementaryTransactionKind}(x) \\ \vee\ & \mathsf{EntityType}(x) \\ \vee\ & \mathsf{CompositeEntityType}(x)\end{aligned} \tag{E.50}$$ FMcs

$$\begin{aligned}\mathsf{FactRelation}(\langle s,t\rangle) \implies & \mathsf{attribute}(s,t) \\ \vee\ & \mathsf{generalisation}(s,t) \\ \vee\ & \mathsf{specialisation}(s,t) \\ \vee\ & \mathsf{aggregation}(s,t)\end{aligned} \tag{E.51}$$ FMrel

OFD

$$\mathsf{ofdDiagram}(d) \implies \mathsf{DiagramType}(d) \tag{E.52}$$ OFDd

$$\begin{aligned}\mathsf{ofdDiagramElement}(x,d) \implies & \mathsf{FactConcept}(x) \\ \wedge\ & \mathsf{ofdDiagram}(d)\end{aligned} \tag{E.53}$$ OFDe

$$\begin{aligned}\mathsf{ofdDiagramRelation}(r,d) \implies & \mathsf{FactRelation}(r) \\ \wedge\ & \mathsf{ofdDiagram}(d)\end{aligned} \tag{E.54}$$ OFDr

$$\begin{aligned}\mathsf{ofdDiagramRelation}(\langle s,t\rangle) \implies & \mathsf{adomain}(s,t) \\ \vee\ & \mathsf{aggregation}(s,t) \\ \vee\ & \mathsf{arange}(s,t) \\ \vee\ & \mathsf{attribute}(s,t) \\ \vee\ & \mathsf{concern}(s,t) \\ \vee\ & \mathsf{event}(s,t) \\ \vee\ & \mathsf{generalisation}(s,t) \\ \vee\ & \mathsf{pdomain}(s,t) \\ \vee\ & \mathsf{prange}(s,t) \\ \vee\ & \mathsf{relation}(s,t) \\ \vee\ & \mathsf{specialisation}(s,t)\end{aligned} \tag{E.55}$$ OFDrel

ET

$$\mathsf{concern}(x, y) \implies \mathsf{ProductKind}(x) \wedge \mathsf{ElementaryTransactionKind}(y) \qquad (\mathrm{E}.56)\ \text{PKconTK}$$

$$\mathsf{attribute}(x, y) \implies \mathsf{adomain}(x) \wedge \mathsf{arange}(y) \qquad (\mathrm{E}.57)\ \text{AT}$$

$$\mathsf{relation}(x, y) \implies \mathsf{pdomain}(x) \wedge \mathsf{prange}(y) \qquad (\mathrm{E}.58)\ \text{ATrel}$$

$$\mathsf{FTcontainedinTK}(x, y) \implies \mathsf{FactType}(x) \wedge \mathsf{TransactionKind}(y) \qquad (\mathrm{E}.59)\ \text{FTinTK}$$

$$\exists_x \left[\mathsf{baseEntity}(x) \implies \mathsf{event}(x)\right] \qquad (\mathrm{E}.60)\ \text{BE}$$

$$\mathsf{event}(x) \implies \mathsf{baseEntity}(x) \qquad (\mathrm{E}.61)\ \text{Event}$$

$$\begin{aligned} \mathsf{property1}(x, y) &\implies \neg\mathsf{property2}(x, y) \\ \mathsf{property2}(x, y) &\implies \neg\mathsf{property1}(x, y) \end{aligned} \qquad (\mathrm{E}.62)\ \text{PT}$$

E.5 Action Metamodel

$$\mathsf{ActionModel}(x) \implies \mathsf{ActionConcept}(x) \lor \mathsf{ActionRelation}(x) \tag{E.63}$$ AM

$$\mathsf{ActionConcept}(x) \implies \mathsf{TransactionProcessStepKind}(x) \lor \mathsf{ActionRule}(x) \tag{E.64}$$ AMcs

$$\mathsf{ActionRelation}(\langle s, t \rangle) \implies \mathsf{when}(s,t) \lor \mathsf{while}(s,t) \lor \mathsf{with}(s,t) \lor \mathsf{trigger}(s,t) \tag{E.65}$$ AMrel

ARD

$$\mathsf{ardDiagram}(d) \implies \mathsf{DiagramType}(d) \tag{E.66}$$ ARDd

$$\mathsf{ardDiagramElement}(x,d) \implies \mathsf{ActionConcept}(x) \land \mathsf{ardDiagram}(d) \tag{E.67}$$ ARDi

$$\mathsf{ardDiagramRelation}(r,d) \implies \mathsf{ActionRelation}(r) \land \mathsf{ardDiagram}(d) \tag{E.68}$$ ARDr

$$\mathsf{ardDiagramRelation}(\langle s, t \rangle) \implies \mathsf{when}(s,t) \lor \mathsf{while}(s,t) \lor \mathsf{with}(s,t) \lor \mathsf{trigger}(s,t) \tag{E.69}$$ ARDrel

Appendix F

Verification rules programmed

F.1 Construction Metamodel

The propositions of a specific type are implicit programmed in the verification because all List<>s are of the right type.

```
public List<TransactionKind> TransactionKinds { get; set; } = new List<
    TransactionKind>();
```

Listing F.1: Elementary Transaction kind

```
public List<AggregateTransactionKind> AggregateTransactionKinds { get; set;
    } = new List<AggregateTransactionKind>();
```

Listing F.2: Aggregate Transaction kind

```
public List<ElementaryActorRole> ElementaryActorRoles { get; set; } = new
    List<ElementaryActorRole>();
```

Listing F.3: Elementary Actor Role

```
public List<CompositeActorRole> CompositeActorRoles { get; set; } = new
    List<CompositeActorRole>();
```

Listing F.4: Composite Actor Role

The formulas explicitly written in the eq. E.6 and eq. E.5 are implicit covered by building the lists as shown in listing F.5 and listing F.6, respectively.

$$\begin{aligned} \mathsf{TransactionKind}(x) \implies & \mathsf{ElementaryTransactionKind}(x) \\ & \veebar \mathsf{AggregateTransactionKind}(x) \end{aligned} \qquad \text{(E.6 ref)}_{\text{TK}}$$

$$\begin{aligned} \mathsf{ActorRole}(x) \implies & \mathsf{ElementaryActorRole}(x) \\ & \veebar \mathsf{CompositeActorRole}(x) \end{aligned} \qquad \text{(E.5 ref)}_{\text{AR}}$$

```
transactionkind = new List<BaseKind>();
transactionkind.AddRange(m.TransactionKinds);
transactionkind.AddRange(m.AggregateTransactionKinds);
```

Listing F.5: Transaction kind (see eq. E.6)

```
actorrole = new List<BaseKind>();
actorrole.AddRange(m.ElementaryActorRoles);
actorrole.AddRange(m.CompositeActorRoles);
```

Listing F.6: Actor role (see eq. E.5)

The construction model is build up of actor roles and transactions. This eq. E.2 is programmed in listing F.7.

$$\begin{aligned}\mathsf{ConstructionModel}(x) \implies & \mathsf{ConstructionConcept}(x) \\ & \vee \mathsf{ConstructionRelation}(x)\end{aligned} \qquad \text{(E.2 ref) } \textsc{cm}$$

```
constructionmodel = new List<BaseKind>();
constructionmodel.AddRange(actorrole);
constructionmodel.AddRange(transactionkind);
```

Listing F.7: Construction Model (see eq. E.2)

The mutual exclusion formula of a TK (eq. E.6) and AR (eq. E.5) can be checked by the pieces of program in listing F.8 and listing F.9, respectively.

```
b = null;
if ((b = m.TransactionKinds.Find(etk => m.AggregateTransactionKinds.Any(
    atk => atk.Id == etk.Id))) != null)
{
  ErrMessage("CM002", "ETK␣cannot␣be␣also␣a␣ATK", b);
}
```

Listing F.8: Transaction kind exclusion (see eq. E.6)

```
BaseKind b = null;
if ((b = m.ElementaryActorRoles.Find(ear => m.CompositeActorRoles.Any(car
     => car.Id == ear.Id))) != null)
{
  ErrMessage("CM001", "EAR␣cannot␣be␣also␣a␣CAR", b);
}
```

Listing F.9: Actor role exclusion (see eq. E.5)

$$\mathsf{executor}(x, y) \implies \mathsf{TransactionKind}(x) \wedge \mathsf{ActorRole}(y) \qquad \text{(E.7 ref) } \textsc{tke}$$

```
case Connector.CT.ExecutorEAR:
  if (null == m.TransactionKinds.Find(x => x.Id == c.Source))
  {
    ErrMessage("CN001", "No␣TK␣in␣ExecutorEAR␣as␣source", c);
  }
  if (null == m.ElementaryActorRoles.Find(x => x.Id == c.Target))
  {
    ErrMessage("CN002", "No␣EAR␣in␣ExecutorEAR␣as␣target", c);
  }
  break;
case Connector.CT.ExecutorCAR:
  if (null == m.TransactionKinds.Find(x => x.Id == c.Source))
  {
    ErrMessage("CN011", "No␣TK␣in␣ExecutorCAR␣as␣source", c);
  }
  if (null == m.CompositeActorRoles.Find(x => x.Id == c.Target))
  {
    ErrMessage("CN012", "No␣CAR␣in␣ExecutorCAR␣as␣target", c);
  }
  break;
```

Listing F.10: Executor role check (see eq. E.7)

$$\mathsf{initiator}(x, y) \implies \mathsf{ActorRole}(x) \wedge \mathsf{TransactionKind}(y) \qquad \text{(E.8 ref)}_{\text{TKi}}$$

```
case Connector.CT.InitiatorEAR:
  if (null == m.ElementaryActorRoles.Find(x => x.Id == c.Source))
  {
    ErrMessage("CN003", "No␣EAR␣in␣InitiatorEAR␣as␣source", c);
  }
  if (null == m.TransactionKinds.Find(x => x.Id == c.Target))
  {
    ErrMessage("CN004", "No␣TK␣in␣InitiatorEAR␣as␣target", c);
  }
  break;
case Connector.CT.InitiatorCAR:
  if (null == m.CompositeActorRoles.Find(x => x.Id == c.Source))
  {
    ErrMessage("CN005", "No␣CAR␣in␣InitiatorCAR␣as␣source", c);
  }
  if (null == m.TransactionKinds.Find(x => x.Id == c.Target))
  {
    ErrMessage("CN006", "No␣TK␣in␣InitiatorCAR␣as␣target", c);
  }
```

```
        break;
```

Listing F.11: Executor role check (see eq. E.8)

F.2 Action Metamodel

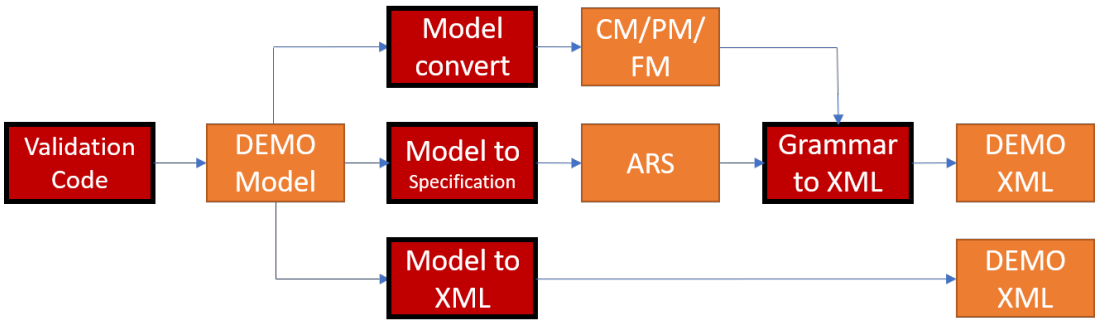


Figure F.1: ARS Test path

F.2.1 Validation Code

```
ActionRuleType ar01 = new ActionRuleType()
{
  Id = Guid.NewGuid(),
  Name = "test",
  Event = ep01,
  Assess = as01,
  Response = rs01
};
m.ActionRuleTypes.Add(ar01);
```

Listing F.12: Action Rule Code Specification

```
EventPartType ep01 = new EventPartType()
{
  Agendum = ag01
};
```

Listing F.13: Action Rule Code Even Part

```
AssessType as01 = new AssessType() { };
```

Listing F.14: Action Rule Code Assess Part

```
IfType rs01 = new IfType()
{
```

```
  Then = new List<BlockType>() { new BlockType() { Action = new
      ActionType() { TransactionProcessStepKind = tpsk02ac.Id } } }
};
```

Listing F.15: Action Rule Code Response Part

```
ActionType ag01 = new ActionType()
{
  TransactionProcessStepKind = tpsk02rq.Id,
  With = wh00
};
```

Listing F.16: Action Rule Code Agendum Clause

```
List<BlockType> wh00 = new List<BlockType>()
  {
    new BlockType(){ AttributeProperty= new FmReferenceType(){ Entity =
        et01.Id, Attribute = at0101.Id } },
    new BlockType(){ AttributeProperty= new FmReferenceType(){ Entity =
        et01.Id, Attribute = at0102.Id } },
    new BlockType(){ AttributeProperty= new FmReferenceType(){ Entity =
        et01.Id, Connection = cn08.Id } }
  };
```

Listing F.17: Action Rule Code When/With Clause

F.2.2 Model Convert

```
Dictionary<string, string> Transactions = m.TransactionKinds.Select(p =>
    new { Key = p.Identification, Value = p.Id.ToString() }).ToDictionary(
    x => x.Key, x => x.Value);
```

Listing F.18: Model convert CM

```
Dictionary<string, string> TransactionSteps = new Dictionary<string,
    string>();
foreach (TransactionProcessStepKind tpsk in m.TransactionProcessStepKinds
    )
{
  TransactionKind tk = m.TransactionKinds.Find(x => x.Id == tpsk.
      transactionKind);
  string abbr = ModelStructure.Constants.Element.TPSK.
      TransactionStateAbbreviations.Find(x => x.Item1 == tpsk.state).Item2
      ;
  TransactionSteps.Add(tk.Identification + "/" + abbr, tpsk.Id.ToString()
      );
}
```

Listing F.19: Model convert PM

```
Dictionary<string, string> Entities = m.EntityTypes.Select(p => new { Key
    = p.Identification, Value = p.Id.ToString() }).ToDictionary(x => x.
    Key, x => x.Value);

Dictionary<string, string> Attributes = new Dictionary<string, string>();
foreach (AttributeType at in m.AttributeTypes)
{
  EntityType et = m.EntityTypes.Find(x => x.Id == at.EntityType);
  Attributes.Add(et.Identification + "/" + at.Identification, at.Id.
      ToString());
}

Dictionary<string, string> Connections = new Dictionary<string, string>()
    ;
foreach (Connector cn in m.Connections)
{
  if (cn.ConnectorType == Connector.CT.ReferenceET)
  {
    EntityType et = m.EntityTypes.Find(x => x.Id == cn.Source);
    Connections.Add(et.Identification + "/" + cn.Name, cn.Id.ToString());
  }
}
```

Listing F.20: Model convert FM

F.2.3 Model to Specification

```
public static StringBuilder action_rule_specification(MSM.ActionRuleType ar
    )
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.RuleKeyword));
  sb.Append("␣");
  sb.Append(ar.Name);
  sb.Append("\r\n");
  indent += 2;
  sb.Append(event_part(ar.Event));
  sb.Append(assess_part(ar.Assess));
  sb.Append(response_part(ar.Event, ar.Response));
  indent -= 2;
  return sb;
}
```

Listing F.21: Action Rule Creation Specification

```
static private StringBuilder event_part(MSM.EventPartType ev)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(agendum_clause(ev.Agendum));
  if (null != ev.While)
    sb.Append(while_clause(ev.While));
  return sb;
}
```

Listing F.22: Action Rule Creation Even Part

```
private static StringBuilder assess_part(MSM.AssessType bt)
{
  StringBuilder sb = new StringBuilder();
  if (null != bt && (null != bt.Justice || null != bt.Sincerity || null !=
      bt.Truth))
  {
    sb.Append("".PadLeft(indent));
    sb.Append(Keyword(keywords.AccessKeyword));
    sb.Append("\r\n");

    indent += 2;
    //sb.Append(justice_sub_part());
    //sb.Append(sincerity_sub_part());
    //truth_sub_part();
    indent -= 2;
  }
  return sb;
}

private static StringBuilder justice_sub_part(List<MSM.FmReferenceType> rts
    )
{
  StringBuilder sb = new StringBuilder();
  sb.Append("".PadLeft(indent));
  sb.Append(Keyword(keywords.JusticeKeyword));
  sb.Append(":");
  sb.Append("\r\n");
  indent += 2;
  foreach (MSM.FmReferenceType rt in rts)
  {
    sb.Append(fact_kind_formulation(rt));
  }
  indent -= 2;
  return sb;
}
```

```
private static StringBuilder sincerity_sub_part(List<MSM.FmReferenceType>
    rts)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.SincirityKeyword));
  sb.Append(":");
  sb.Append("\r\n");
  indent += 2;
  foreach (MSM.FmReferenceType rt in rts)
  {
    sb.Append(fact_kind_formulation(rt));
  }
  indent -= 2;
  return sb;
}

private static StringBuilder truth_sub_part(List<MSM.FmReferenceType> rts)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.TruthKeyword));
  sb.Append(":");
  sb.Append("\r\n");
  indent += 2;
  sb.Append(foreach_clause(null));
  sb.Append("{");
  foreach (MSM.FmReferenceType rt in rts)
  {
    sb.Append(fact_kind_formulation(rt));
  }
  sb.Append("}");
  foreach (MSM.FmReferenceType rt in rts)
  {
    sb.Append(fact_kind_formulation(rt));
  }
  indent -= 2;
  return sb;
}
```

Listing F.23: Action Rule Creation Assess Part

```
static private StringBuilder response_part(MSM.EventPartType ev, MSM.IfType
    it)
{
  StringBuilder sb = new StringBuilder();
  MSM.TransactionProcessStepKind tpsk = _m.TransactionProcessStepKinds.Find
      (x => x.Id == ev.Agendum.TransactionProcessStepKind);
  sb.Append("".PadLeft(indent));
```

```
  sb.Append(Keyword(keywords.IfKeyword));
  sb.Append("␣");
  sb.Append(Keyword(keywords.ComplyingKeyword));
  sb.Append("␣");
  sb.Append(Keyword(keywords.WithKeyword));
  sb.Append("␣");
  sb.Append(present_tense_intention(tpsk.state));
  sb.Append("␣");
  sb.Append(Keyword(keywords.IsKeyword));
  sb.Append("␣");
  sb.Append(Keyword(keywords.ConsideredKeyword));
  sb.Append("␣");
  sb.Append(Keyword(keywords.JustifiableKeyword));
  sb.Append("\r\n");
  sb.Append("".PadLeft(indent));
  sb.Append(Keyword(keywords.ThenKeyword));
  sb.Append("\r\n");
  indent += 2;
  foreach (MSM.BlockType bt in it.Then)
  {
    if (null != bt.Flow && null != bt.Flow.Foreach)
    {
      sb.Append("".PadLeft(indent));
      sb.Append(foreach_clause(bt.Flow.Foreach));
      sb.Append("{");
      foreach (MSM.BlockType bt2 in bt.Flow.Foreach.Repeating)
      {
        sb.Append(action_clause(bt2));
      }
      sb.Append("}");
    }
    else
    {
      sb.Append(action_clause(bt));
    }
  }
  if (null != it.Else && it.Else.Count > 0)
  {
    sb.Append(Keyword(keywords.ElseKeyword));
    foreach (MSM.BlockType bt in it.Else)
    {
      sb.Append(action_clause(bt));
    }
  }
  indent -= 2;
  return sb;
}
```

```
static private StringBuilder action_clause(MSM.BlockType bt)
{
  StringBuilder sb = new StringBuilder();
  MSM.TransactionProcessStepKind tpsk = _m.TransactionProcessStepKinds.Find
      (x => x.Id == bt.Action.TransactionProcessStepKind);
  MSM.TransactionKind tk = _m.TransactionKinds.Find(x => x.Id == tpsk.
      transactionKind);

  Guid etId = _m.Connections.Find(x => x.ConnectorType == MSM.Connector.CT.
      ConcernsETTK && x.Target == tk.Id).Source;
  string etName = _m.EntityTypes.Find(x => x.Id == etId).Identification;

  sb.Append("".PadLeft(indent));
  sb.Append(present_tense_intention(tpsk.state));
  sb.Append("␣");
  sb.Append(transaction_reference(tk.Name, etName));
  sb.Append("␣");
  sb.Append(transaction_step_reference(tk, tpsk));
  sb.Append(with_clause(bt.Action.With));
  return sb;
}
```

Listing F.24: Action Rule Creation Response Part

```
private static StringBuilder agendum_clause(ModelStructure.ActionType ag)
{
  StringBuilder sb = new StringBuilder();
  sb.Append("".PadLeft(indent));
  sb.Append(Keyword(keywords.WhenKeyword));
  sb.Append("␣");

  MSM.TransactionProcessStepKind tpsk = _m.TransactionProcessStepKinds.Find
      (x => x.Id == ag.TransactionProcessStepKind);
  MSM.TransactionKind tk = _m.TransactionKinds.Find(x => x.Id == tpsk.
      transactionKind);

  Guid etId = _m.Connections.Find(x => x.ConnectorType == MSM.Connector.CT.
      ConcernsETTK && x.Target == tk.Id).Source;
  string etName = _m.EntityTypes.Find(x => x.Id == etId).Identification;

  sb.Append(transaction_reference(tk.Name, etName));
  sb.Append("␣");
  sb.Append(Keyword(keywords.IsKeyword));
  sb.Append("␣");
  sb.Append(perfect_tense_intention(tpsk.state));
  sb.Append("␣");
```

```
  sb.Append(transaction_step_reference(tk, tpsk));
  sb.Append("\r\n");
  sb.Append(with_clause(ag.With));
  return sb;
}
```

Listing F.25: Action Rule Creation Agendum Clause

```
private static StringBuilder with_clause(List<MSM.BlockType> bts)
{
  StringBuilder sb = new StringBuilder();
  if (null != bts && bts.Count > 0)
  {
    sb.Append("".PadLeft(indent));
    sb.Append(Keyword(keywords.WithKeyword));
    sb.Append("\r\n");
    indent += 2;
    foreach (MSM.BlockType bt in bts)
    {
      sb.Append("".PadLeft(indent));
      sb.Append(property_kind_formulation(bt.AttributeProperty));
      sb.Append("\r\n");
    }
    indent -= 2;
  }
  return sb;
}

private static StringBuilder property_kind_formulation(MSM.FmReferenceType
    rt)
{
  StringBuilder sb = new StringBuilder();
  MSM.AttributeType at = _m.AttributeTypes.Find(x => x.Id == rt.Attribute);
  MSM.EntityType et = _m.EntityTypes.Find(x => x.Id == rt.Entity);
  MSM.Connector cn = _m.Connections.Find(x => x.ConnectorType == MSM.
      Connector.CT.ReferenceET && x.Id == rt.Connection);
  MSM.EntityType cnet = _m.EntityTypes.Find(x => x.Id == cn?.Source);
  string atName = "";
  string etName = "";
  string atType = "";
  if (rt.Connection == new Guid())
  {
    atName = at.Identification;
    etName = et.Identification;
    atType = at.TypeName;
  }
  else
```

```
  {
    atName = cn.Name;
    etName = cnet.Identification; ;
    atType = et.Identification;
  }

  sb.Append(property_variable(atName, etName, ""));
  sb.Append(Keyword(keywords.IsKeyword));
  sb.Append("␣");
  int q = 1;
  switch (q)
  {
    case 1:
      // If the attribute is a relation, some is needed
      if (rt.Connection != new Guid())
      {
        sb.Append(Keyword(keywords.SomeKeyword));
        sb.Append("␣");
      }
      sb.Append(bracketed_name(atType));
      break;
    case 2:
      property_variable(at.Identification, et.Identification, "");
      break;
    case 3:
      sb.Append(Keyword(keywords.SetKeyword));
      sb.Append("␣");
      sb.Append(Keyword(keywords.OfKeyword));
      sb.Append("␣");
      bracketed_name("");
      sb.Append("\r\n");
      sb.Append("{");
      List<MSM.BlockType> rts = new List<MSM.BlockType>();
      with_clause(rts);
      sb.Append("}");
      break;
  }
  sb.Append(";");
  return sb;
}
```

Listing F.26: Action Rule Creation With Clause

```
static private StringBuilder foreach_clause(MSM.ForeachType fet)
{
  StringBuilder sb = new StringBuilder();
  sb.Append("".PadLeft(indent));
```

```
  sb.Append(Keyword(keywords.ForKeyword));
  sb.Append("␣");
  sb.Append(Keyword(keywords.EachKeyword));
  sb.Append("␣");
  bracketed_name(fet.Element);
  sb.Append("␣");
  sb.Append(Keyword(keywords.InKeyword));
  sb.Append("␣");
  sb.Append("{");
  property_variable("", "", "");
  sb.Append("}");
  return sb;
}
```

Listing F.27: Action Rule Creation For Each Clause

```
private static StringBuilder while_clause(ModelStructure.BlockType bl)
{
  StringBuilder sb = new StringBuilder();
  sb.Append("".PadLeft(indent));
  sb.Append(Keyword(keywords.WhileKeyword));
  foreach_clause(null);
  sb.Append("{");
  while_subclause();
  sb.Append("}");
  while_subclause();
  return sb;
}
private static StringBuilder while_subclause()
{
  StringBuilder sb = new StringBuilder();
  transaction_reference(null, null);
  sb.Append(Keyword(keywords.IsKeyword));
  perfect_tense_intention(0);
  transaction_step_reference(null, null);
  with_clause(null);
  return sb;
}

private static StringBuilder transaction_step_reference(MSM.TransactionKind
    tk, MSM.TransactionProcessStepKind tpsk)
{
  StringBuilder sb = new StringBuilder();
  sb.Append("(");
  sb.Append(tk.Identification);
  sb.Append("/");
  sb.Append(intention_abbreviation(tpsk.state));
```

```
  sb.Append(")");
  return sb;
}

private static StringBuilder transaction_reference(string tkName, string
    etName)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(bracketed_name(tkName));
  sb.Append("␣");
  sb.Append(Keyword(keywords.ForKeyword));
  sb.Append("␣");
  sb.Append(Keyword(keywords.NewKeyword));
  sb.Append("␣");
  sb.Append(bracketed_name(etName));
  return sb;
}
```

Listing F.28: Action Rule Creation While Clause

```
private static StringBuilder property_variable(string atName, string etName
    , string period)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.TheKeyword));
  sb.Append("␣");
  sb.Append(bracketed_name(atName));
  sb.Append("␣");
  sb.Append(Keyword(keywords.OfKeyword));
  sb.Append("␣");
  sb.Append(bracketed_name(etName));
  sb.Append("␣");
  if (!string.IsNullOrEmpty(period))
  {
    sb.Append(Keyword(keywords.InKeyword));
    sb.Append("␣");
    sb.Append(bracketed_name(period));
  }
  return sb;
}
```

Listing F.29: Action Rule Creation Property Variable

```
private static StringBuilder fact_kind_formulation(MSM.FmReferenceType rt)
{
  StringBuilder sb = new StringBuilder();
  if (sb == null) // checkout proper if
  {
```

```
    sb.Append(property_kind_formulation(rt));
  }
  else
  {
    sb.Append(attribute_kind_formulation());
  }
  return sb;
}
```

Listing F.30: Action Rule Creation Fact Kind

```
private static StringBuilder attribute_kind_formulation()
{
  StringBuilder sb = new StringBuilder();
  property_variable("", "", "");
  sb.Append(Keyword(keywords.IsKeyword));
  comparison();
  value_variable();
  sb.Append(";");
  return sb;
}

private static StringBuilder comparison()
{
  StringBuilder sb = new StringBuilder();
  operators();
  sb.Append(Keyword(keywords.CompareOrKeyword));

  comparison();
  operators();
  sb.Append(Keyword(keywords.CompareAndKeyword));

  comparison();
  operators();
  return sb;
}

private static StringBuilder operators()
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.CompareEqualKeyword));
  sb.Append(Keyword(keywords.CompareToKeyword));
  sb.Append(Keyword(keywords.CompareGreaterKeyword));
  sb.Append(Keyword(keywords.CompareThanKeyword));
  sb.Append(Keyword(keywords.CompareLessKeyword));
  sb.Append(Keyword(keywords.CompareThanKeyword));
  sb.Append(Keyword(keywords.InKeyword));
```

```
  return sb;
}

private static StringBuilder value_variable()
{
  StringBuilder sb = new StringBuilder();
  property_variable("", "", "");
  value();
  ;

  return sb;
}

private static StringBuilder value()
{
  StringBuilder sb = new StringBuilder();
  dimension("");
  sb.Append(":");
  //Real()
  // Integer
  unit("");
  boolean(false);
  return sb;
}

private static StringBuilder dimension(string name)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(name);
  return sb;
}

private static StringBuilder boolean(bool b)
{
  StringBuilder sb = new StringBuilder();
  if (b)
    sb.Append(Keyword(keywords.TrueKeyword));
  else
    sb.Append(Keyword(keywords.FalseKeyword));
  return sb;
}

private static StringBuilder unit(string name)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(name);
  return sb;
```

```
}
```

Listing F.31: Action Rule Creation Attribute Kind

```
private static string present_tense_intention(ModelStructure.Constants.
    Element.TPSK.TransactionStates st)
{
  return st.ToString().ToLower();
}

private static string intention_abbreviation(ModelStructure.Constants.
    Element.TPSK.TransactionStates st)
{
  return ModelStructure.Constants.Element.TPSK.
      TransactionStateAbbreviations.Find(x => x.Item1 == st).Item2;
}

private static string perfect_tense_intention(ModelStructure.Constants.
    Element.TPSK.TransactionStates st)
{
  return ModelStructure.Constants.Element.TPSK.TransactionStatePerfectTense
      .Find(x => x.Item1 == st).Item2.ToLower();
}
```

Listing F.32: Action Rule Creation Intentions

```
public static StringBuilder stringed_name(string name)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.DoubleQuote));
  sb.Append(name);
  sb.Append(Keyword(keywords.DoubleQuote));
  return sb;
}

public static StringBuilder bracketed_name(string name)
{
  StringBuilder sb = new StringBuilder();
  sb.Append(Keyword(keywords.BracketPre));
  sb.Append(name);
  sb.Append(Keyword(keywords.BracketPost));
  return sb;
}

enum keywords
{
  SomeKeyword,
  RuleKeyword,
```

```
    WhenKeyword,
    WhileKeyword,
    WithKeyword,
    IsKeyword,
    ForKeyword,
    EachKeyword,
    NewKeyword,
    OfKeyword,
    InKeyword,
    OnKeyword,
    TheKeyword,
    IfKeyword,
    ElseKeyword,
    ThenKeyword,
    SetKeyword,

    RequestKeyword,
    PromiseKeyword,
    StateKeyword,
    AcceptKeyword,
    DeclineKeyword,
    QuitKeyword,
    RejectKeyword,
    StopKeyword,
    RevokeKeyword,
    AllowKeyword,
    RefuseKeyword,
    ExecuteKeyword,

    RequestedKeyword,
    PromisedKeyword,
    StatedKeyword,
    AcceptedKeyword,
    DeclinedKeyword,
    QuittedKeyword,
    RejectedKeyword,
    StoppedKeyword,
    RevokedKeyword,
    AllowedKeyword,
    RefusedKeyword,
    ExecutedKeyword,

    RequestAbKeyword,
    PromiseAbKeyword,
    StateAbKeyword,
    AcceptAbKeyword,
    DeclineAbKeyword,
```

```
  QuitAbKeyword,
  RejectAbKeyword,
  StopAbKeyword,
  RevokeAbKeyword,
  AllowAbKeyword,
  RefuseAbKeyword,
  ExecuteAbKeyword,

  TrueKeyword,
  FalseKeyword,


  AccessKeyword,
  JusticeKeyword,
  SincirityKeyword,
  TruthKeyword,

  CompareEqualKeyword,
  CompareGreaterKeyword,
  CompareLessKeyword,
  CompareThanKeyword,
  CompareToKeyword,
  CompareOrKeyword,
  CompareAndKeyword,
  DoubleQuote,
  BracketPre,
  BracketPost,
  LineEndKeyword,
  ComplyingKeyword,
  ConsideredKeyword,
  JustifiableKeyword
}

static string Keyword(keywords kw)
{
  string word = "";
  switch (kw)
  {

    case keywords.SomeKeyword: word = "some"; break;
    case keywords.RuleKeyword: word = "rule"; break;
    case keywords.WhenKeyword: word = "when"; break;
    case keywords.WhileKeyword: word = "while"; break;
    case keywords.WithKeyword: word = "with"; break;
    case keywords.IsKeyword: word = "is"; break;
    case keywords.ForKeyword: word = "for"; break;
    case keywords.EachKeyword: word = "each"; break;
```

```
case keywords.NewKeyword: word = "new"; break;
case keywords.OfKeyword: word = "of"; break;
case keywords.InKeyword: word = "in"; break;
case keywords.OnKeyword: word = "on"; break;
case keywords.TheKeyword: word = "the"; break;
case keywords.IfKeyword: word = "if"; break;
case keywords.ElseKeyword: word = "else"; break;
case keywords.ThenKeyword: word = "then"; break;
case keywords.SetKeyword: word = "set"; break;

case keywords.RequestKeyword: word = "request"; break;
case keywords.PromiseKeyword: word = "promise"; break;
case keywords.StateKeyword: word = "state"; break;
case keywords.AcceptKeyword: word = "accept"; break;
case keywords.DeclineKeyword: word = "decline"; break;
case keywords.QuitKeyword: word = "quit"; break;
case keywords.RejectKeyword: word = "reject"; break;
case keywords.StopKeyword: word = "stop"; break;
case keywords.RevokeKeyword: word = "revoke"; break;
case keywords.AllowKeyword: word = "allow"; break;
case keywords.RefuseKeyword: word = "refuse"; break;
case keywords.ExecuteKeyword: word = "execute"; break;

case keywords.RequestedKeyword: word = "requested"; break;
case keywords.PromisedKeyword: word = "promised"; break;
case keywords.StatedKeyword: word = "stated"; break;
case keywords.AcceptedKeyword: word = "accepted"; break;
case keywords.DeclinedKeyword: word = "declined"; break;
case keywords.QuittedKeyword: word = "quitted"; break;
case keywords.RejectedKeyword: word = "rejected"; break;
case keywords.StoppedKeyword: word = "stopped"; break;
case keywords.RevokedKeyword: word = "revoked"; break;
case keywords.AllowedKeyword: word = "allowed"; break;
case keywords.RefusedKeyword: word = "refused"; break;
case keywords.ExecutedKeyword: word = "executed"; break;

case keywords.RequestAbKeyword: word = "rq"; break;
case keywords.PromiseAbKeyword: word = "pm"; break;
case keywords.StateAbKeyword: word = "st"; break;
case keywords.AcceptAbKeyword: word = "ac"; break;
case keywords.DeclineAbKeyword: word = "dc"; break;
case keywords.QuitAbKeyword: word = "qt"; break;
case keywords.RejectAbKeyword: word = "rj"; break;
case keywords.StopAbKeyword: word = "sp"; break;
case keywords.RevokeAbKeyword: word = "rv"; break;
case keywords.AllowAbKeyword: word = "al"; break;
case keywords.RefuseAbKeyword: word = "rf"; break;
```

```
        case keywords.ExecuteAbKeyword: word = "ex"; break;

        case keywords.TrueKeyword: word = "true"; break;
        case keywords.FalseKeyword: word = "false"; break;

        case keywords.AccessKeyword: word = "assess"; break;
        case keywords.JusticeKeyword: word = "justice"; break;
        case keywords.SincirityKeyword: word = "sincerity"; break;
        case keywords.TruthKeyword: word = "truth"; break;

        case keywords.CompareEqualKeyword: word = "equal"; break;
        case keywords.CompareGreaterKeyword: word = "greater"; break;
        case keywords.CompareLessKeyword: word = "less"; break;
        case keywords.CompareThanKeyword: word = "than"; break;
        case keywords.CompareToKeyword: word = "to"; break;
        case keywords.CompareOrKeyword: word = "or"; break;
        case keywords.CompareAndKeyword: word = "and"; break;
        case keywords.DoubleQuote: word = "\""; break;
␣␣␣␣␣␣␣␣case␣keywords.BracketPre:␣word␣=␣"[";␣break;
␣␣␣␣␣␣␣␣case␣keywords.BracketPost:␣word␣=␣"]";␣break;
␣␣␣␣␣␣␣␣case␣keywords.LineEndKeyword:␣word␣=␣";";␣break;
␣␣␣␣␣␣␣␣case␣keywords.ComplyingKeyword:␣word␣=␣"complying";␣break;
␣␣␣␣␣␣␣␣case␣keywords.ConsideredKeyword:␣word␣=␣"considered";␣break;
␣␣␣␣␣␣␣␣case␣keywords.JustifiableKeyword:␣word␣=␣"justifiable";␣break;

␣␣␣␣␣␣}
␣␣␣␣␣␣return␣word;
␣␣␣␣}
␣␣␣␣
```

Listing F.33: Action Rule Creation Variables

```
rule test
  when [order baking] for new [Pizza] is requested (T99-002/rq)
  with
    the [name] of [Pizza] is [TEKST];
    the [price] of [Pizza] is [EURO];
    the [pizzaorder] of [Customer] is some [Pizza];
  if complying with request is considered justifiable
  then
    accept [order baking] for new [Pizza] (T99-002/ac)
```

Figure F.2: ARS Test Grammar Out

F.2.4 Grammar to XML

```
rule test when [order baking ] for new [Pizza ] is (T99-002/rq)
  with the [name ] of [Pizza ] is [TEKST ];
the [price ] of [Pizza ] is [EURO ];
the [pizzaorder ] of [Customer ] is  some [Pizza ];
if complying with request is considered justifiable
then
accept [order baking ] for new [Pizza ](T99-002/ac)
```

Figure F.3: ARS Test Grammar In

```
public override void EnterAction_rule_specification([NotNull]
    AMgrammarPropParser.Action_rule_specificationContext context)
{
  m = new AMParser.XML.DEMOmodel() { Name = "AM" };
  m.ActionRuleTypes = new AMParser.XML.ActionRuleType[1];
  m.ActionRuleTypes[0] = new AMParser.XML.ActionRuleType()
  {
    Id = Guid.NewGuid().ToString(),
    Name = ((AMgrammarPropParser.Bracketed_nameContext)context.children[1]).
        Start.Text
  };
}
```

Listing F.34: Action Rule to XML Specification

```
public override void EnterAgendum_clause([NotNull] AMgrammarPropParser.
    Agendum_clauseContext context)
{
  string intention = context.transaction_step_reference().
      intention_abbreviation().Start.Text;
  string tk = context.transaction_step_reference().transaction_kind_id().
      Start.Text;
  string tpskId = TransactionSteps[tk + "/" + intention];
  m.ActionRuleTypes[0].Event = new AMParser.XML.EventPartType()
  {
    Agendum = new AMParser.XML.ActionType()
    {
      TransactionProcessStepKind = tpskId
    }
  };

  if (context.with_clause().Length > 0)
  {
    int withCount = context.with_clause()[0].ChildCount;
    if (withCount > 0)
```

```
    {
      m.ActionRuleTypes[0].Event.Agendum.With = new AMParser.XML.BlockType[
          withCount - 1];
      for (int i = 0; i < withCount - 1; i++)
      {
        m.ActionRuleTypes[0].Event.Agendum.With[i] = new AMParser.XML.
            BlockType() { };
      }
    }
  }
}
```

Listing F.35: Action Rule to XML Agendum Clause

```
public override void EnterProperty_kind_formulation([NotNull]
    AMgrammarPropParser.Property_kind_formulationContext context)
{
  AMgrammarPropParser.Agendum_clauseContext ag = context.Parent.Parent as
      AMgrammarPropParser.Agendum_clauseContext;
  AMgrammarPropParser.With_clauseContext wt = context.Parent as
      AMgrammarPropParser.With_clauseContext;
  if (ag != null)
  {
    string att = ((AMgrammarPropParser.NameContext)context.property_variable
        ()[0].bracketed_name()[0].children[1]).Start.Text;
    string ent = ((AMgrammarPropParser.NameContext)context.property_variable
        ()[0].bracketed_name()[1].children[1]).Start.Text;
    string attId = null;
    string entId = Entities[ent];
    string conId = null;
    int i = 0;
    for (int j = 1; j < wt.children.Count; j++)
    {
      if (wt.children[j] == context)
      {
        i = j - 1;
        break;
      }
    }
    if (context.children[2].ToString() == "some") // Connection
    {
      conId = Connections[ent + "/" + att];
    }
    else
    {
      attId = Attributes[ent + "/" + att];
    }
```

```
    m.ActionRuleTypes[0].Event.Agendum.With[i].Item = new FmReferenceType()
    {
      Attribute = attId,
      Entity = entId,
      Connection = conId
    };
  }
}
```

Listing F.36: Action Rule to XML With Clause

```
<ActionRuleTypes>
  <ActionRuleType Id="ee7f8d87-b370-4a9f-ab47-09e2b335549c">
    <Name>test</Name>
    <Event>
      <Agendum>
        <TransactionProcessStepKind>8d2d7f8d-4357-4840-837e-284725f9a1b7</TransactionProcessStepKind>
        <With>
          <AttributeProperty>
            <Entity>407fa8cf-13e3-40f7-ab8a-c61b8bd7ca77</Entity>
            <Attribute>109fe0be-5f86-44d1-9e1d-e5947f65eb3f</Attribute>
          </AttributeProperty>
        </With>
        <With>
          <AttributeProperty>
            <Entity>407fa8cf-13e3-40f7-ab8a-c61b8bd7ca77</Entity>
            <Attribute>3645c18d-7836-46d1-af60-e5e0461b4f38</Attribute>
          </AttributeProperty>
        </With>
        <With>
          <AttributeProperty>
            <Entity>a765d01e-22d8-4a96-8d6d-9282a5d5a8fc</Entity>
            <Connection>b0cf1db2-c2a1-45c1-bd67-aa8f91cfd33f</Connection>
          </AttributeProperty>
        </With>
      </Agendum>
    </Event>
  </ActionRuleType>
```

Figure F.4: ARS Test XML Out

F.2.5 Model to XML

```
public static ActionRuleType ConvertToXML(MSM.ActionRuleType ART)
{
  return new ActionRuleType
  {
    Id = ART?.Id.ToString(),
    Name = ART?.Name,
    Event = ConvertToXML(ART?.Event),
    Assess = ConvertToXML(ART?.Assess),
    Response = ConvertToXML(ART?.Response)
  };
}
```

Listing F.37: Action Rule Model to XML Specification

```
public static EventPartType ConvertToXML(MSM.EventPartType ET)
{
  return new EventPartType
  {
    Agendum = ConvertToXML(ET?.Agendum),
    While = (ET?.While != null) ? ConvertToXML(ET?.While) : null
  };
}
```

Listing F.38: Action Rule Model to XML Even Part

```
public static AssessPartType ConvertToXML(MSM.AssessType AT)
{
  return new AssessPartType
  {
    Justice = (AT?.Justice?.Count > 0) ? AT?.Justice?.Select(x =>
        ConvertToXML(x)).ToArray() : null,
    Sincerity = (AT?.Sincerity?.Count > 0) ? AT?.Sincerity?.Select(x =>
        ConvertToXML(x)).ToArray() : null,
    Truth = ConvertToXML(AT?.Truth)
  };
}
```

Listing F.39: Action Rule Model to XML Assess Part

```
public static ForeachType ConvertToXML(MSM.ForeachType FET)
{
  return new ForeachType
  {
    Element = FET?.Element,
    Collection = FET?.Collection.ToString(),
    Repeating = (FET?.Repeating.Count > 0) ? FET?.Repeating.Select(x =>
        ConvertToXML(x)).ToArray() : null
  };
}
```

Listing F.40: Action Rule Model to XML For Each Clause

```
<ActionRuleTypes>
  <ActionRuleType Id="4e915e16-8ecf-4fa6-911c-43894890cef4">
    <Name>test</Name>
    <Event>
      <Agendum>
        <TransactionProcessStepKind>8d2d7f8d-4357-4840-837e-284725f9a1b7</TransactionProcessStepKind>
        <With>
          <AttributeProperty>
            <Entity>407fa8cf-13e3-40f7-ab8a-c61b8bd7ca77</Entity>
            <Attribute>109fe0be-5f86-44d1-9e1d-e5947f65eb3f</Attribute>
          </AttributeProperty>
        </With>
        <With>
          <AttributeProperty>
            <Entity>407fa8cf-13e3-40f7-ab8a-c61b8bd7ca77</Entity>
            <Attribute>3645c18d-7836-46d1-af60-e5e0461b4f38</Attribute>
          </AttributeProperty>
        </With>
        <With>
          <AttributeProperty>
            <Entity>407fa8cf-13e3-40f7-ab8a-c61b8bd7ca77</Entity>
            <Connection>b0cf1db2-c2a1-45c1-bd67-aa8f91cfd33f</Connection>
          </AttributeProperty>
        </With>
      </Agendum>
    </Event>
```

Figure F.5: ARS Test XML DEMO

Appendix G

SEA implementation Metamodel

We did the implementation of the MDG in SEA with this browser tree of fig. G.1. The leafs represent the profile, diagrams and toolbox.

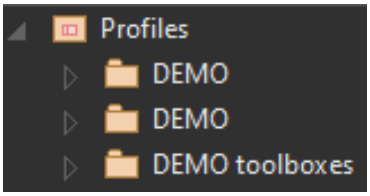


Figure G.1: SEA browser tree

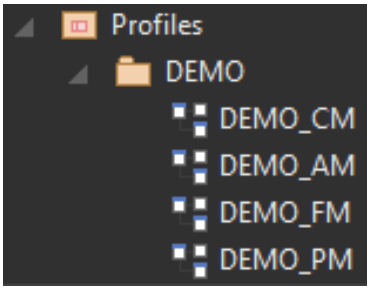


Figure G.2: SEA browser aspect tree

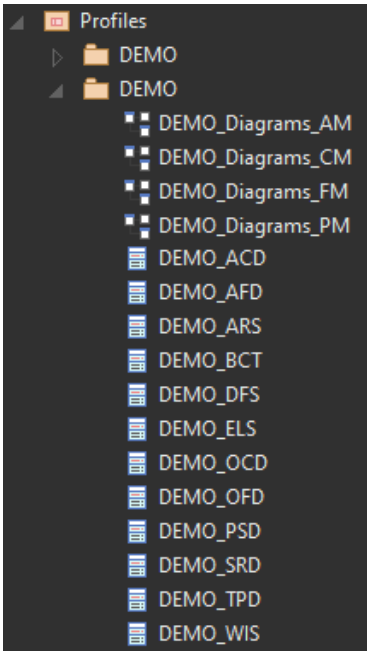


Figure G.3: SEA browser diagram tree

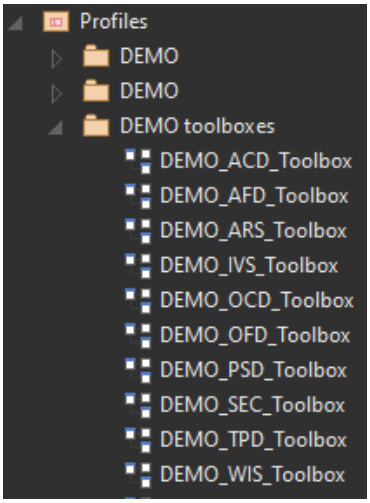


Figure G.4: SEA browser diagram tree

G.1 Profile

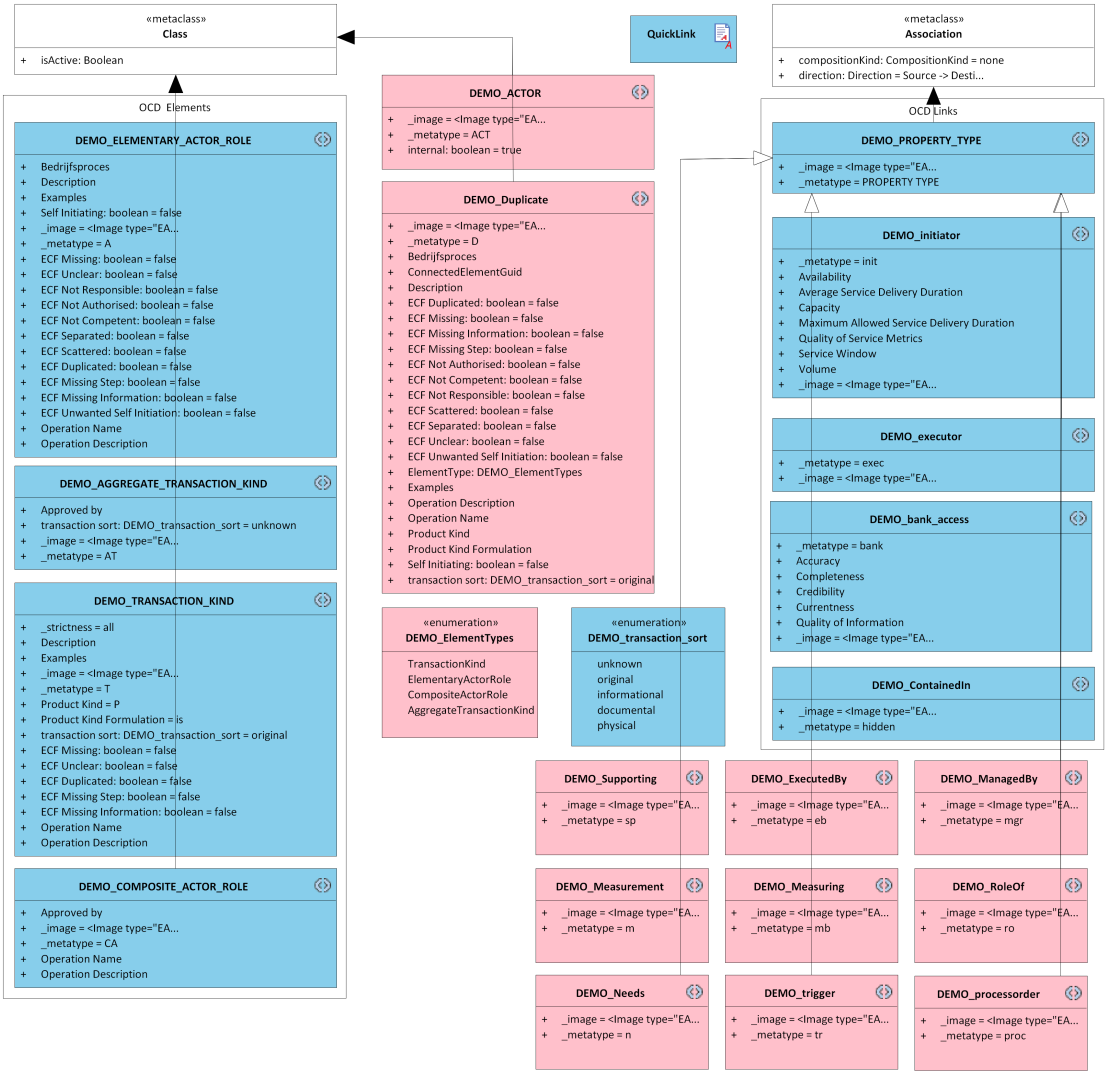


Figure G.5: SEA CM profile

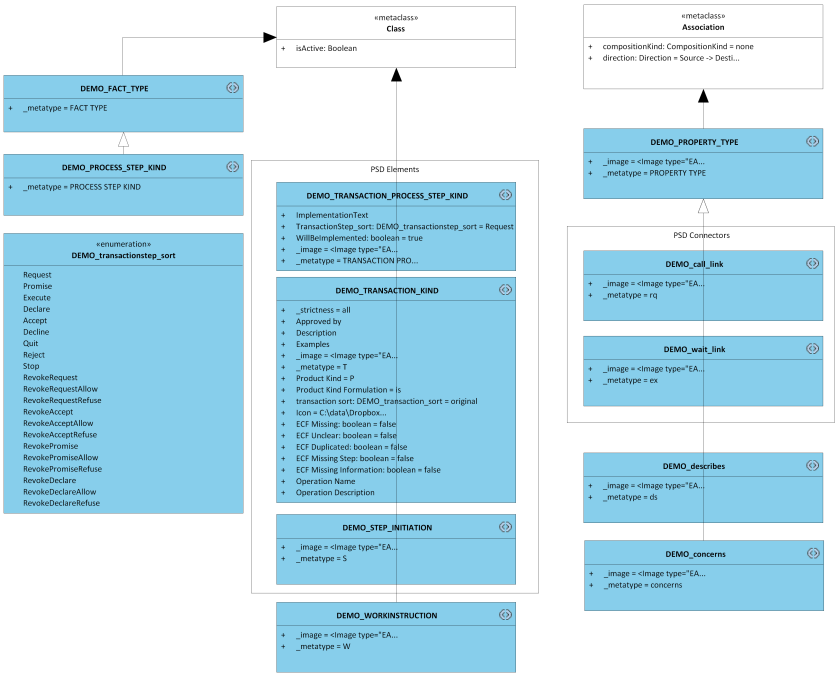


Figure G.6: SEA PM profile

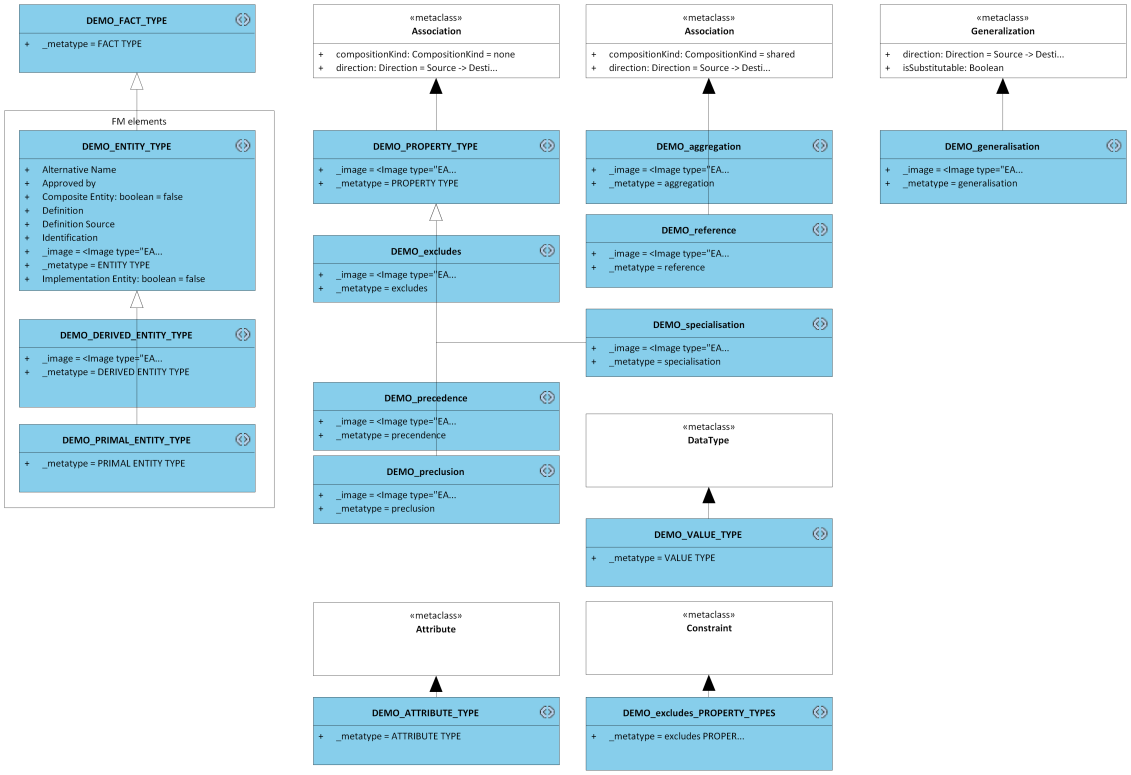
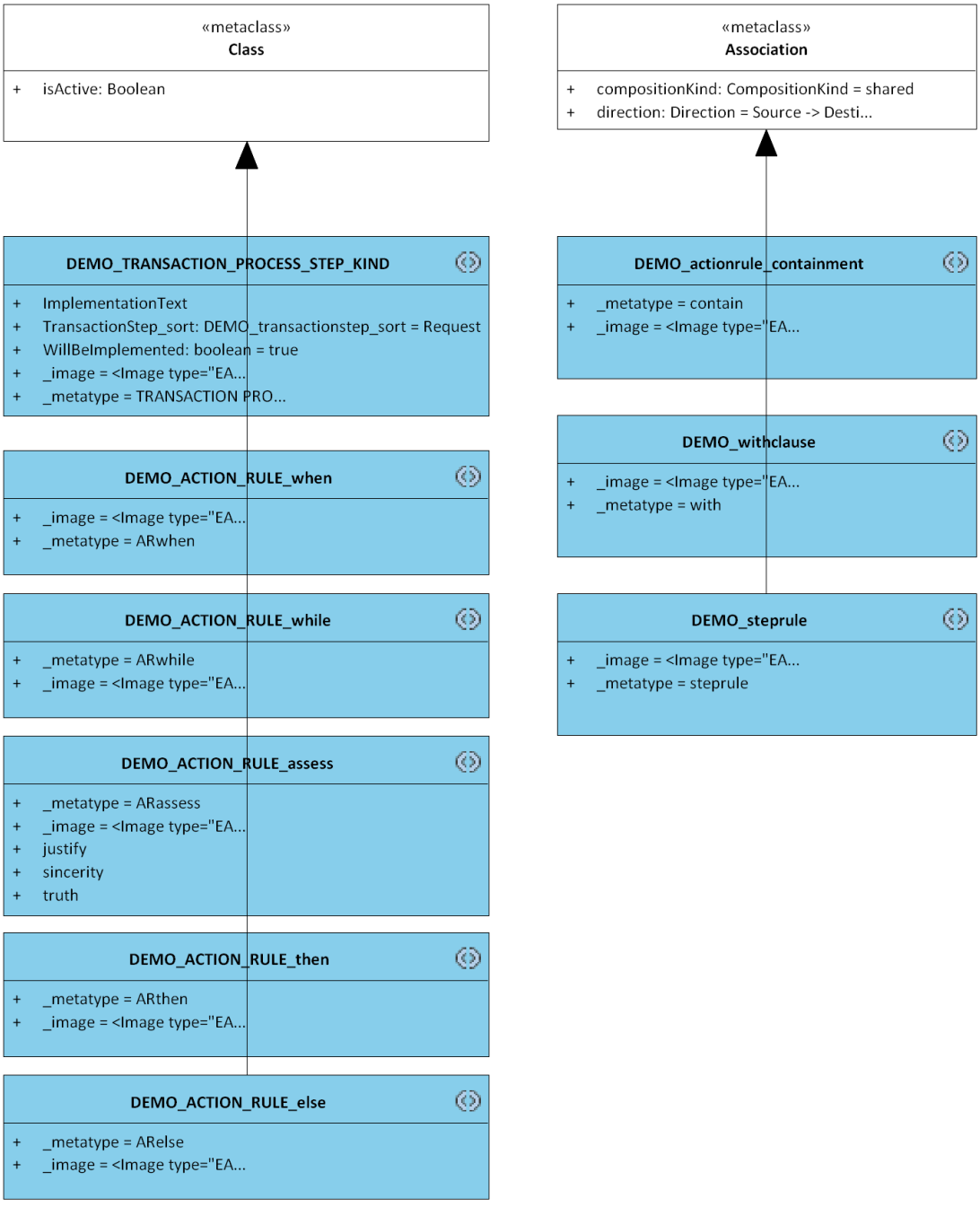


Figure G.7: SEA FM profile


«metaclass»
Class
+ isActive: Boolean
«metaclass»
Association
+ compositionKind: CompositionKind = shared
+ direction: Direction = Source -> Desti...
DEMO_TRANSACTION_PROCESS_STEP_KIND
+ ImplementationText
+ TransactionStep_sort: DEMO_transactionstep_sort = Request
+ WillBeImplemented: boolean = true
+ _image = <Image type="EA...
+ _metatype = TRANSACTION PRO...
DEMO_ACTION_RULE_when
+ _image = <Image type="EA...
+ _metatype = ARwhen
DEMO_ACTION_RULE_while
+ _metatype = ARwhile
+ _image = <Image type="EA...
DEMO_ACTION_RULE_assess
+ _metatype = ARassess
+ _image = <Image type="EA...
+ justify
+ sincerity
+ truth
DEMO_ACTION_RULE_then
+ _metatype = ARthen
+ _image = <Image type="EA...
DEMO_ACTION_RULE_else
+ _metatype = ARelse
+ _image = <Image type="EA...
DEMO_actionrule_containment
+ _metatype = contain
+ _image = <Image type="EA...
DEMO_withclause
+ _image = <Image type="EA...
+ _metatype = with
DEMO_steprule
+ _image = <Image type="EA...
+ _metatype = steprule

G.2 Toolbox

Figure G.9: SEA ACD Toolbox

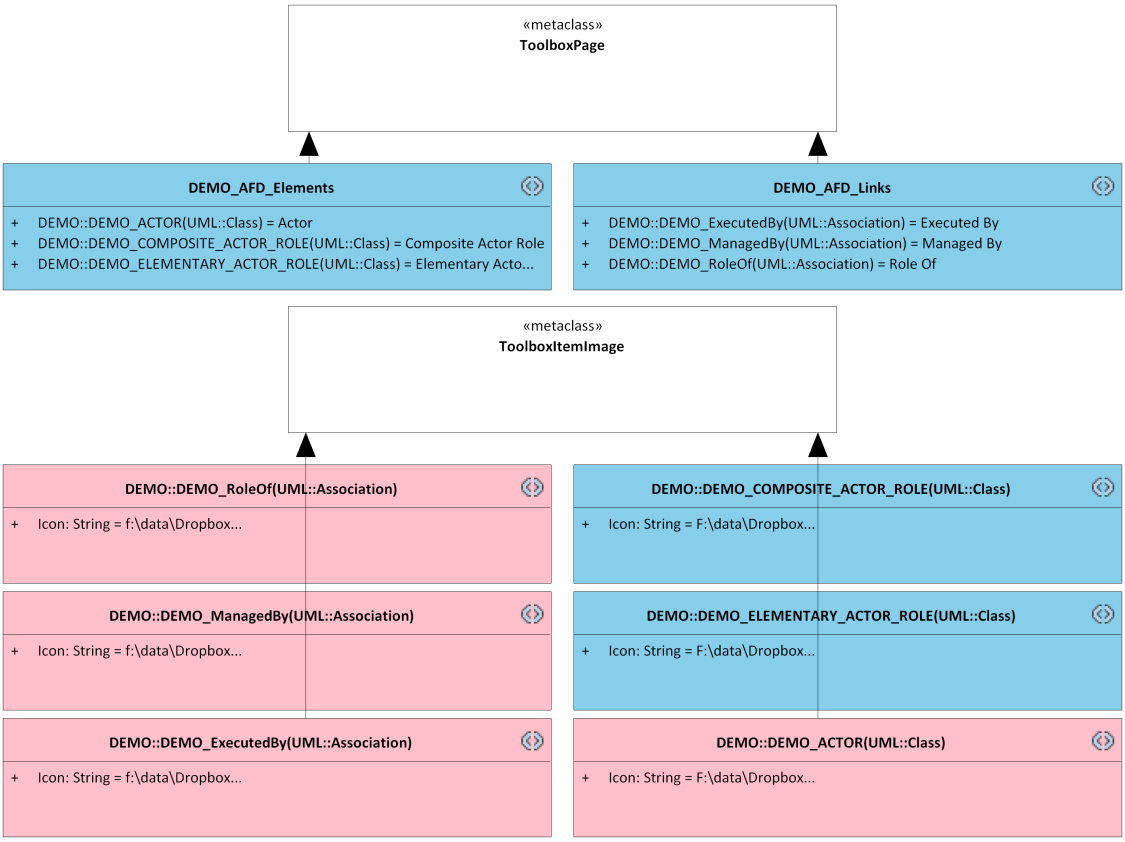


Figure G.10: SEA AFD Toolbox

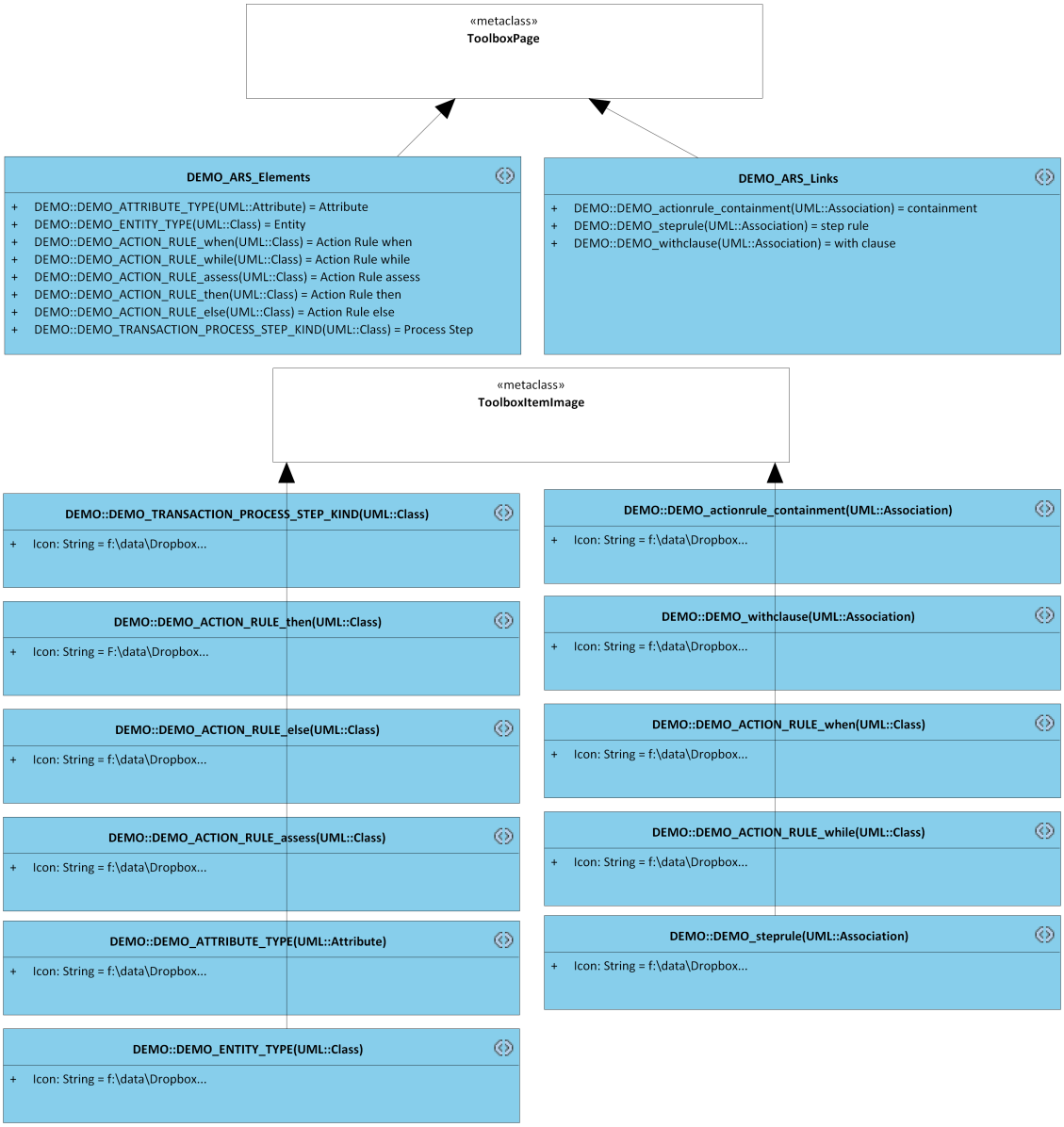


Figure G.11: SEA ARS Toolbox

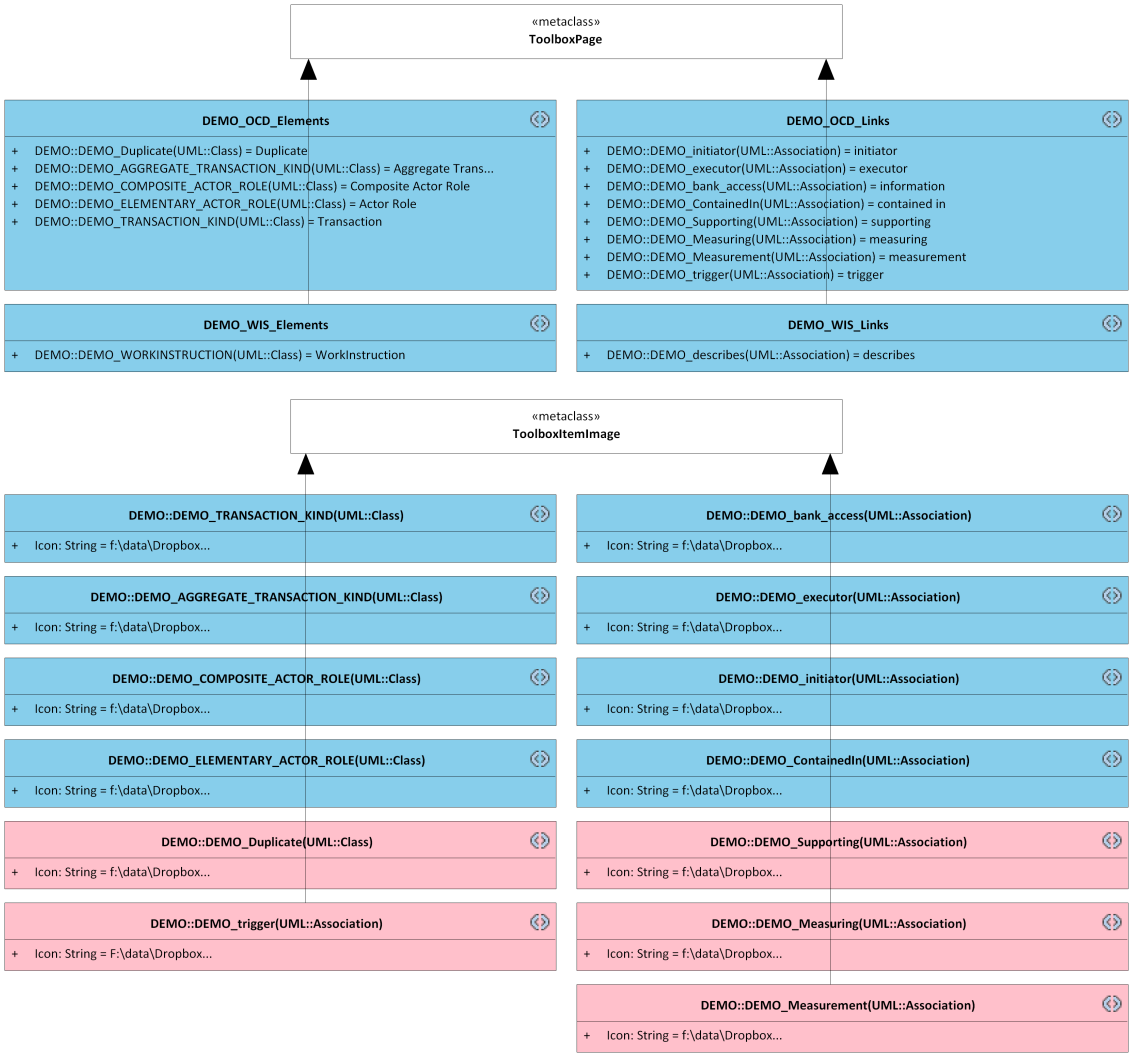


Figure G.12: SEA OCD Toolbox

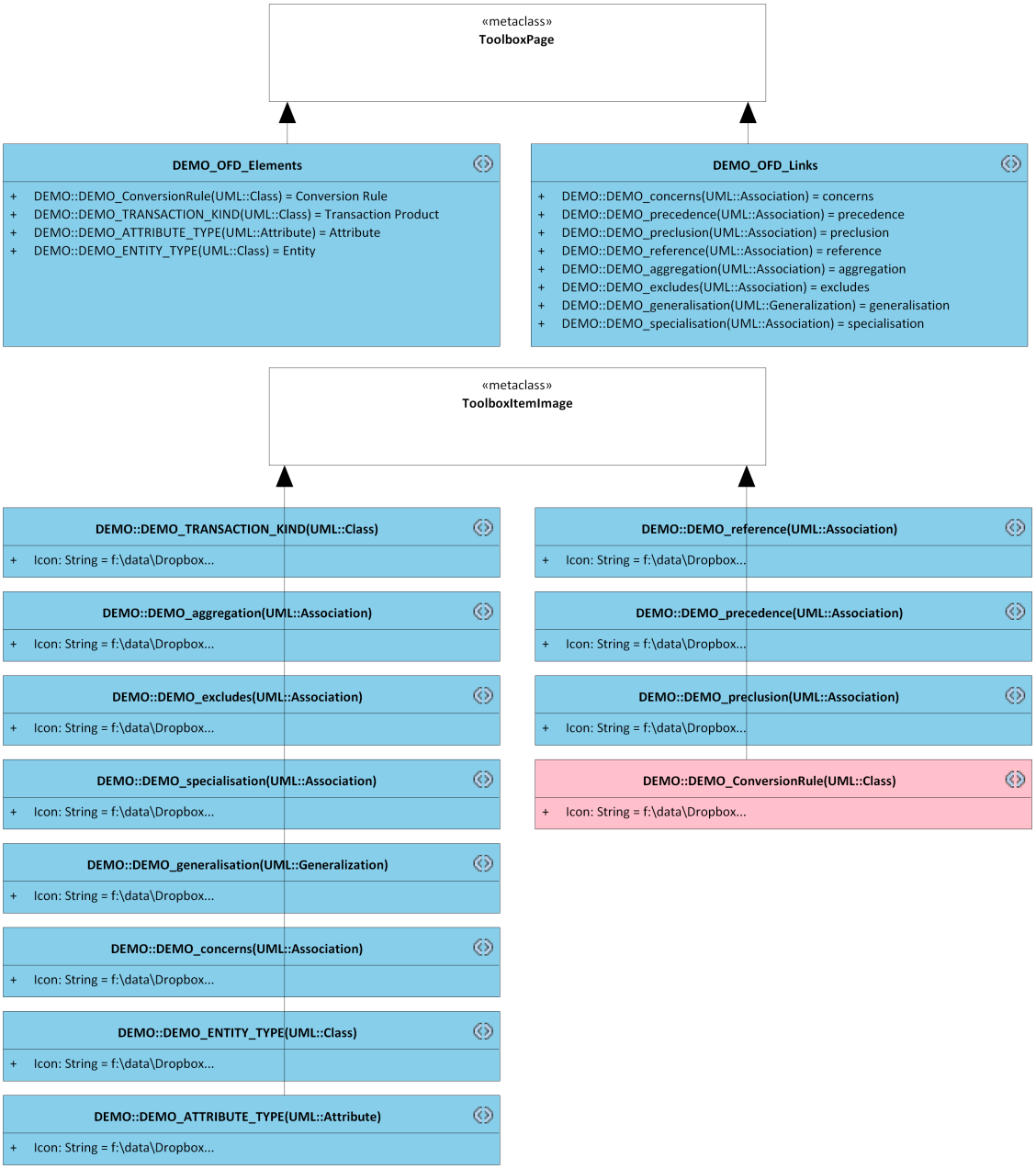


Figure G.13: SEA OFD Toolbox

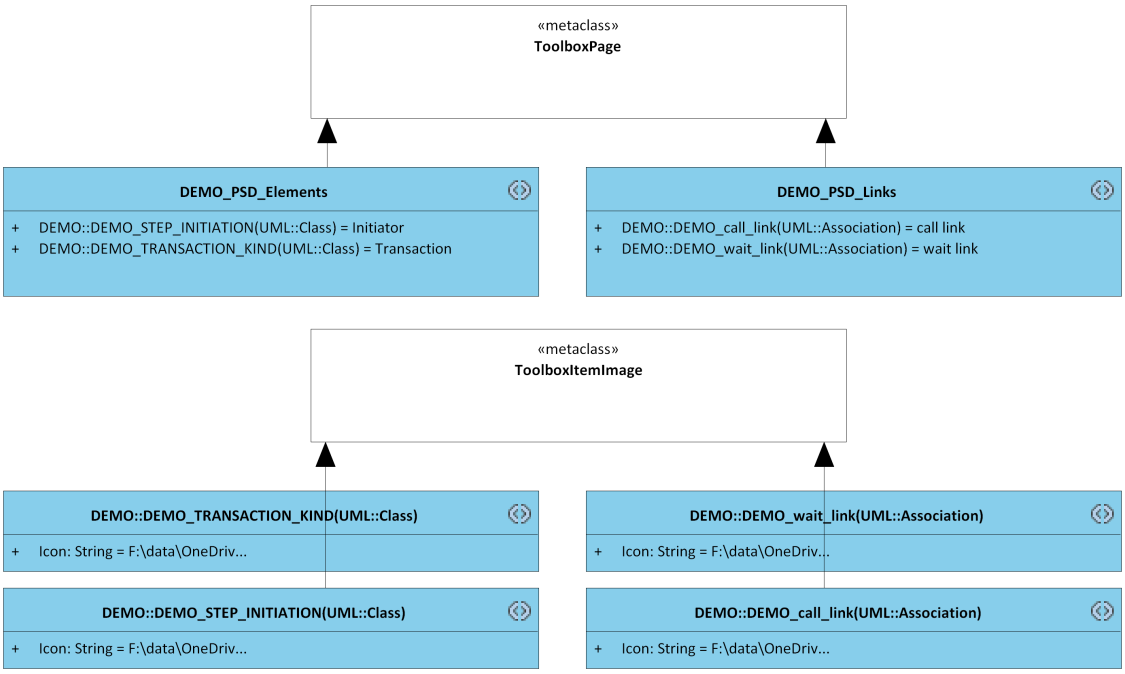


Figure G.14: SEA PSD Toolbox

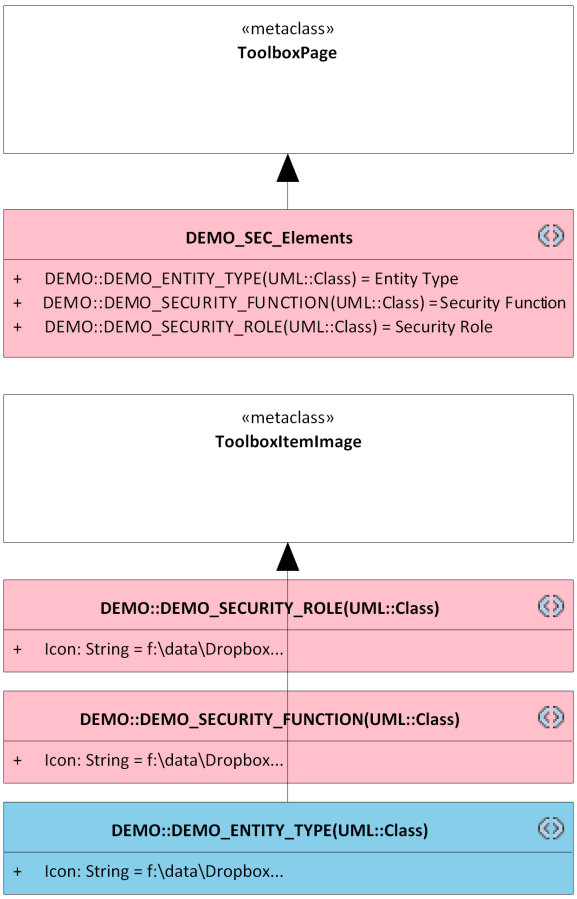


Figure G.15: SEA SEC Toolbox

Figure G.16: SEA TPD Toolbox

Figure G.17: SEA WIS Toolbox

G.3 Diagrams

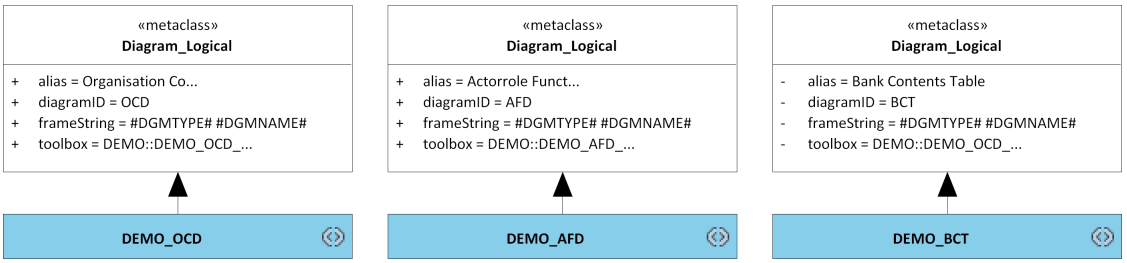


Figure G.18: SEA CM diagrams

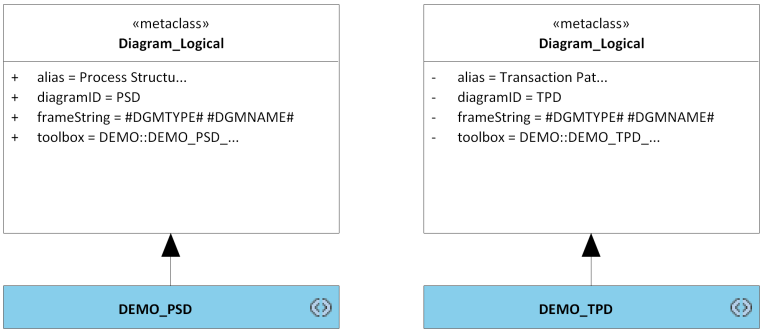


Figure G.19: SEA PM diagrams

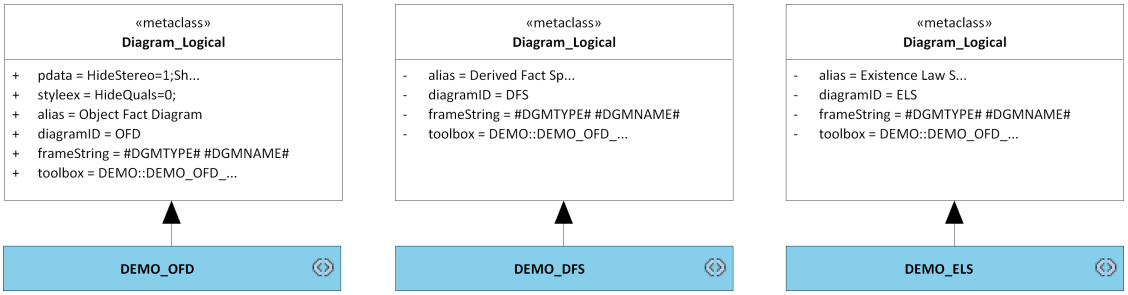


Figure G.20: SEA FM diagrams

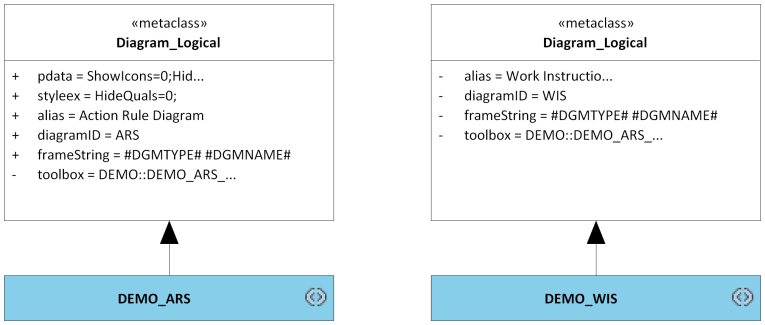


Figure G.21: SEA AM diagrams

G.4 Shapes

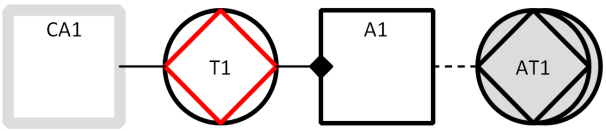


Figure G.22: DEMOSL CM OCD visualisation

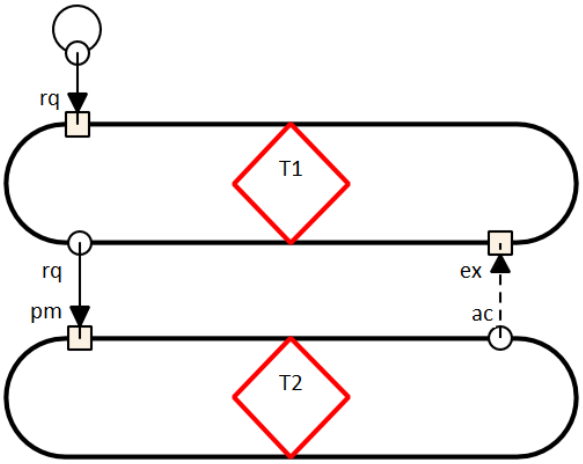


Figure G.23: DEMOSL PSD visualisation proposal

G.5 ShapeScipts

Shapescripts of the whole tool solution have been listed here. The ordering is by usage for standard DEMO and extra functionality.

G.5.1 Elements

```
// ShapeScript StereoType: DEMO_TRANSACTION_KIND

shape main
{
        layoutType = "leftright";
        DefSize(50,50);

        if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OCD"))
        {
                AddSubShape("OCDTransactionKind", 100, 100, 0, 0);
        }
    else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
        AddSubShape("PSDTransactionKind", 100, 100, 0, 0);
    }
    else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OFD"))
    {
        AddSubShape("OFDTransactionKind", 100, 100, 0, 0);
    }
    else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_TPD"))
    {
        AddSubShape("TPDTransactionKind", 100, 100, 0, 0);
    }
    else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_SRD"))
    {
        AddSubShape("SRDTransactionKind", 100, 100, 0, 0);
    }
    else if (hasproperty("diagram.mdgtype", "ArchiMate3::Business"))
    {
        AddSubShape("FNCTransactionKind", 100, 100, 0, 0);
    }
    else {
     AddSubShape("OCDTransactionKind", 100, 100, 0, 0);
     //print("#diagram.mdgtype#");
    }

shape FNCTransactionKind
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
        fixedAspectRatio = "true";
        DefSize(100,50);
        setpenwidth(2);
        if (HasTag("transaction␣sort", "original"))
```

```
	{
		SetPenColor(255, 0, 0);
	}
	else if (HasTag("transaction␣sort", "informational"))
	{
		SetPenColor(0, 255, 0);
	}
	else if (HasTag("transaction␣sort", "documental"))
	{
		SetPenColor(0, 0, 255);
	}
	else if (HasTag("transaction␣sort", "fysical"))
	{
		SetPenColor(128, 128, 128);
	}
	StartPath();
	moveto(0,0);
	lineto(75,0);
	lineto(75,-25);
	lineto(100,50);
	lineto(75,125);
	lineto(75,100);
	lineto(0,100);
	lineto(0,0);
	EndPath();
	setfillcolor(255,255,192);
	FillAndStrokePath();
	print("#tag:Operation␣Name#");
}

shape OFDTransactionKind
{
	h_align = "center";
	v_align = "center";
	editablefield = "name";
	fixedAspectRatio = "true";
	setpenwidth(2);
	//SetFillColor(255, 255, 255);
	//roundrect(0, 0, 100, 100, 30, 30);
	//SetPenColor(0, 0, 0);
	if (HasTag("transaction␣sort", "original"))
	{
		SetPenColor(255, 0, 0);
	}
	else if (HasTag("transaction␣sort", "informational"))
	{
		SetPenColor(0, 255, 0);
```

```
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
                SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
                SetPenColor(128, 128, 128);
        }
        polygon(50, 50, 4, 50, 0);
        print("#tag:Product␣Kind#");
}

shape SRDTransactionKind
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
        fixedAspectRatio = "true";
        setpenwidth(2);
        SetFillColor(127, 127, 127);

        SetPenColor(0, 0, 0);
        polygon(50, 50, 4, 71, 45);
        SetFillColor(191, 191, 191);
        polygon(50, 50, 4, 50, 0);
        print("#tag:Product␣Kind#");
}

shape OCDTransactionKind
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
        fixedAspectRatio = "true";
        setpenwidth(2);
        SetFillColor(255, 255, 255);

        if (HasTag("ECF␣Missing", "true"))
        {
                setlinestyle("dash");
        }
        SetPenColor(0, 0, 0);
        Ellipse(0, 0, 100, 100);
        setlinestyle("solid");

        if (HasTag("transaction␣sort", "original"))
```

```
        {
                SetPenColor(255, 0, 0);
        }
        else if (HasTag("transaction␣sort", "informational"))
        {
                SetPenColor(0, 255, 0);
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
                SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
                SetPenColor(128, 128, 128);
        }
        polygon(50, 50, 4, 50, 0);
        print("#name#");
}

shape PSDTransactionKind
{
        defsize(200, 40);
        setpenwidth(2);
        roundrect(0, 0, 100, 100, 100, 100);
        fillandstrokepath();
        addsubshape("ex", 20, 100, 40, 0);
}

shape TPDTransactionKind
{
        defsize(50, 50);
        setpenwidth(2);
        roundrect(0, 0, 100, 100, 100, 100);
        fillandstrokepath();
        addsubshape("ex", 20, 100, 40, 0);
}

shape ex
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
        setpenwidth(2);
        println("#name#");
    if (HasTag("transaction␣sort", "original"))
        {
                SetPenColor(255, 0, 0);
```

```
        }
        else if (HasTag("transaction␣sort", "informational"))
        {
                SetPenColor(0, 255, 0);
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
                SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
                SetPenColor(128, 128, 128);
        }
        polygon(50, 50, 4, 50, 0);
}

}

shape label
{
        editablefield = "alias";
        setpenwidth(2);
        setorigin("s",0,0);
        //print("#alias#");
        if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OCD"))
        {
                if (hasproperty("diagram.stereotype", "DEMO4n")) {
                }
                else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                }
                else {
                        print("#alias#");
                }
        }
         else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
        {
                print("#alias#");
        }
        else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OFD"))
        {
                print("#tag:Product␣Kind␣Formulation#");
        }
        else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_TPD"))
        {
                print("#alias#");
        }
        else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_SRD"))
```

```
        {
                print("#alias#");
        }
        else if (hasproperty("diagram.mdgtype", "ArchiMate3::Business"))
        {
                print("#name#");
        }
        else {
                print("#alias#");
        }
}
```

Listing G.1: Transaction Kind shapescript

```
// ShapeScript StereoType: DEMO_AGGREGATE_TRANSACTION_KIND

shape main
{
    h_align = "center";
    v_align = "center";
    editablefield = "name";
    fixedAspectRatio = true;
    DefSize(50,50);
    setfillcolor(220,220,220);
        setpenwidth(2);

    ellipse(10,0,110,100);
    ellipse(0,0,100,100);
    if (hasproperty("diagram.stereotype", "DEMO4n")) {
        if (HasTag("transaction␣sort", "original"))
        {
            SetPenColor(255, 0, 0);
        }
        else if (HasTag("transaction␣sort", "informational"))
        {
            SetPenColor(0, 255, 0);
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
            SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
            SetPenColor(128, 128, 128);
        }
    }
    else if (hasproperty("diagram.stereotype", "DEMO4c")) {
```

```
        if (HasTag("transaction␣sort", "original"))
        {
            SetPenColor(255, 0, 0);
        }
        else if (HasTag("transaction␣sort", "informational"))
        {
            SetPenColor(0, 255, 0);
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
            SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
            SetPenColor(128, 128, 128);
        }
    }
    polygon(50,50,4,50,0);

    print("#name#");

}

shape label{
    editablefield = "alias";
    setorigin("s",0,0);
    print("#alias#");
}
```

Listing G.2: Aggregate Transaction Kind shapescript

```
// ShapeScript StereoType: DEMO_ELEMENTARY_ACTOR_ROLE

// setlinestyle(string linestyle) // solid, dash, dot, dashdot, dashdotdot

shape main{
    DefSize(50,50);
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
        AddSubShape("PSDSwimlane", 100, 100, 0, 0);
    }
    else {
        AddSubShape("PSDNormal", 100, 100, 0, 0);
    }

    shape PSDSwimlane
    {
        h_align = "left";
```

```
        v_align = "top";
        editablefield = "name";
        DefSize(30,200);
        setpen(220,220,220,3);
        moveto(0,100);
        lineto(100,100);
                if (hasproperty("diagram.stereotype", "DEMO4n")) {
                }
                else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                }
                else {
                        print("#name#");
                }
    }
    shape PSDNormal
    {
        h_align = "center";
        v_align = "top";
        editablefield = "name";
        DefSize(50,50);
                setpenwidth(2);
        // ECF Missing // ECF 1, 10, 20
        if (HasTag("ECF␣Missing", "true"))
        {
           setlinestyle("dash");
        }
        rectangle(0,0,100,100);
        print("#name#");
    }
}

shape label{
    editablefield = "alias";
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
        setorigin("w",0,0);
                if (hasproperty("diagram.stereotype", "DEMO4n")) {
                }
                else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                }
                else {
                        print("#alias#");
                }
    }
    else {
                if (hasproperty("diagram.stereotype", "DEMO4n")) {
                        setorigin("center", 0, 0);
                }
```

```
            else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                    setorigin("center", 0, 0);
            }
            else {
                    setorigin("s", 0, 0);
            }
            print("#alias#");
        }
}

        //decoration ECFMissing // ECF 1, 10, 20
        //{
        // Not an decoration. Is the boundary.
        //}

        // arcto(int left,int top,int right,int bottom,int startingpointx,int
            startingpointy,int endingpointx,int endingpointy)

        decoration ECFUnclear // ECF 2, 11, 21
    {
        orientation = "N";
        if (HasTag("ECF␣Unclear", "true"))
        {
            //centerbottom point
            setfillcolor(255, 0, 0);
            startpath();
            arc(48, 96, 52, 100, 55, 98, 55, 98);
            endpath();
            fillpath();

            // question mark in 3 sections
            setpen(255, 0, 0, 2);
            moveto(50, 90);
            lineto(50, 70);
            arcto(20, 10, 80, 70, 50, 70, 80, 40);
            arcto(20, 10, 80, 70, 80, 40, 20, 40);
        }
    }

        decoration NotResponsible // ECF 3, 12
    {
        orientation = "SW";

        if (HasTag("ECF␣Not␣Responsible", "true"))
        {
            setpen(255, 0, 0, 1);
            // circle
```

```
        ellipse(0, 0, 100, 100);

        // black V
        setpen(0, 0, 0, 1);
        moveto(20, 20);
        lineto(50, 80);
        lineto(80, 20);
    }
    else if (HasTag("ECF␣Not␣Authorised", "true"))
    {
        // black V
        setpen(0, 0, 0, 1);
        moveto(20, 20);
        lineto(50, 80);
        lineto(80, 20);
    }
    else if (HasTag("ECF␣Not␣Competent", "true"))
    {
        // black V
        setpen(0, 0, 0, 1);
        moveto(20, 20);
        lineto(50, 80);
        lineto(80, 20);
    }
}

    decoration NotQualified // ECF 4, 13
{
    orientation = "S";
    if (HasTag("ECF␣Not␣Authorised", "true"))
    {
        setpen(255, 0, 0, 1);
        // circle
        ellipse(0, 0, 100, 100);
        // black B
        setpen(0, 0, 0, 1);
        moveto(35, 20);
        lineto(35, 80);
        lineto(50, 80);
        arcto(30, 50, 70, 80, 50, 80, 70, 65);
        arcto(30, 50, 70, 80, 70, 65, 50, 50);
        arcto(30, 20, 70, 50, 50, 50, 70, 35);
        arcto(30, 20, 70, 50, 70, 35, 50, 20);
        lineto(35, 20);
        moveto(35, 50);
        lineto(50, 50);
    }
```

```
    else if (HasTag("ECF␣Not␣Responsible", "true"))
    {
        // black B
        setpen(0, 0, 0, 1);
        moveto(35, 20);
        lineto(35, 80);
        lineto(50, 80);
        arcto(30, 50, 70, 80, 50, 80, 70, 65);
        arcto(30, 50, 70, 80, 70, 65, 50, 50);
        arcto(30, 20, 70, 50, 50, 50, 70, 35);
        arcto(30, 20, 70, 50, 70, 35, 50, 20);
        lineto(35, 20);
        moveto(35, 50);
        lineto(50, 50);
    }
    else if (HasTag("ECF␣Not␣Competent", "true"))
    {
        // black B
        setpen(0, 0, 0, 1);
        moveto(35, 20);
        lineto(35, 80);
        lineto(50, 80);
        arcto(30, 50, 70, 80, 50, 80, 70, 65);
        arcto(30, 50, 70, 80, 70, 65, 50, 50);
        arcto(30, 20, 70, 50, 50, 50, 70, 35);
        arcto(30, 20, 70, 50, 70, 35, 50, 20);
        lineto(35, 20);
        moveto(35, 50);
        lineto(50, 50);
    }
}

    decoration NotCompentent // ECF 5, 14
{
    orientation = "SE";
    if (HasTag("ECF␣Not␣Competent", "true"))
    {
        setpen(255, 0, 0, 1);
        // circle
        ellipse(0, 0, 100, 100);
        // black C
        setpen(0, 0, 0, 1);
        arc(30, 20, 70, 80, 70, 30, 70, 70);
    }
    else if (HasTag("ECF␣Not␣Responsible", "true"))
    {
        // black C
```

```
        setpen(0, 0, 0, 1);
        arc(30, 20, 70, 80, 70, 30, 70, 70);
    }
    else if (HasTag("ECF␣Not␣Authorised", "true"))
    {
        // black C
        setpen(0, 0, 0, 1);
        arc(30, 20, 70, 80, 70, 30, 70, 70);
    }
}

    //rectangle(int left, int top, int right, int bottom)
    decoration Separated // ECF 6, 15
{
    orientation = "NW";
    if (HasTag("ECF␣Separated", "true"))
    {
        setfillcolor(255, 0, 0);
        setlinestyle("dash");
        rectangle(0, 0, 70, 70);
        rectangle(30, 30, 100, 100);
    }
    if (HasTag("Self␣Initiating", "true"))
    {
        SetPenColor(0, 0, 0);
        Ellipse(0, 0, 100, 100);
        polygon(50, 50, 4, 50, 0);
    }
}

    decoration Scattered // ECF 7,16
{
    orientation = "E";
    if (HasTag("ECF␣Scattered", "true"))
    {
        setfillcolor(255, 0, 0);
        rectangle(50, 0, 80, 20);
        rectangle(50, 30, 80, 50);
        rectangle(70, 80, 100, 100);
        rectangle(0, 50, 30, 70);
        rectangle(10, 80, 40, 100);
    }
}

    decoration Duplicated // ECF 8, 17, 23
{
    orientation = "NE";
```

```
        if (HasTag("ECF␣Duplicated", "true"))
        {
            setfillcolor(255, 0, 0);
            rectangle(0, 30, 70, 100);
            rectangle(30, 0, 100, 70);
            fillpath();
        }
    }

        decoration MissingStep // ECF 9, 24, 25
    {
        orientation = "W";
        if (HasTag("ECF␣Missing␣Step", "true"))
        {
            setfillcolor(255, 0, 0);
            rectangle(0, 0, 70, 70);
            fillpath();
            setfillcolor(255, 255, 0);
            setlinestyle("dash");
            rectangle(30, 30, 100, 100);
        }
    }

        decoration MissingInformation // ECF 26, 18, 22
    {

    }

    decoration UnwantedSelfInitiation // ECF 19
    {

    }
```

Listing G.3: Elementary Actor Role shapescript

```
// ShapeScript StereoType: DEMO_COMPOSITE_ACTOR_ROLE

shape main{
        DefSize(50,50);
        if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
                AddSubShape("PSDSwimlane", 100, 100, 0, 0);
        }
        else {
                AddSubShape("CARNormal", 100, 100, 0, 0);
        }

        shape PSDSwimlane
```

```
	{
		h_align = "left";
		v_align = "top";
		editablefield = "name";
		DefSize(30,200);
		setpen(220, 220, 220, 3);
		moveto(0,100);
		lineto(100,100);
		if (hasproperty("diagram.stereotype", "DEMO4n")) {
		}
		else if (hasproperty("diagram.stereotype", "DEMO4c")) {
		}
		else {
			print("#name#");
		}
	}
	shape CARNormal
	{
		h_align = "center";
		v_align = "top";
		editablefield = "name";
		DefSize(50,50);
		setpenwidth(2);
		// ECF Missing // ECF 1, 10, 20
		if (HasTag("ECF␣Missing", "true"))
		{
			setlinestyle("dash");
		}
		if (HasProperty('rectanglenotation', '0'))
		{
			setfillcolor(220, 220, 220);
			rectangle(0, 0, 100, 100);
			// code for iconic representation
		}
		else
		{
			setpen(220, 220, 220, 5);
			rectangle(0, 0, 100, 100);
			// code for rectangular representation
			// e.g. Rectangle or DrawNativeShape
		}
		rectangle(0,0,100,100);
		print("#name#");
	}
}

shape label{
```

```
        editablefield = "alias";
        if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
                setorigin("w",0,0);
                if (hasproperty("diagram.stereotype", "DEMO4n")) {
                }
                else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                }
                else {
                        print("#alias#");
                }
        }
        else {
                if (hasproperty("diagram.stereotype", "DEMO4n")) {
                        setorigin("center", 0, 0);
                }
                else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                        setorigin("center", 0, 0);
                }
                else {
                        setorigin("s", 0, 0);
                }
                print("#alias#");
        }
}
```

Listing G.4: Composite Actor Role shapescript

```
// ShapeScript StereoType: DEMO_STEP_INITIATION

shape main{
        defsize(20,20);
        ellipse(0,0,100,100);
}
```

Listing G.5: Step Initiation shapescript

```
// ShapeScript StereoType: DEMO_TRANSACTION_PROCESS_STEP_KIND

shape main
{
        noshadow = true;
    editablefield = "name";
    h_align = "center";
    v_align = "top";
    print("#name#");

    DefSize(50, 50);
        if(hasproperty("diagram.mdgtype", "DEMO::DEMO_ARS"))
```

```
{
    AddSubShape("ARSStep", 100, 100, 0, 0);
} else if(hasproperty("diagram.mdgtype", "DEMO::DEMO_TPD"))
{
    AddSubShape("TPDStep", 100, 100, 0, 0);
} else {
    print("#diagram.mdgtype#");
}

    shape ARSStep
    {
            setfillcolor(255, 255, 255);
            setpen(0,0,0,1);
            // Happy flow
            if(HasTag("TransactionStep_sort", "Request"))
            {
                    setpen(146,208,80,1); // Green-ish
            }
            if(HasTag("TransactionStep_sort", "Promise"))
            {
                    setpen(146,208,80,1); // Green-ish
            }
            if(HasTag("TransactionStep_sort", "Declare"))
            {
                    setpen(146,208,80,1); // Green-ish
            }
            if(HasTag("TransactionStep_sort", "Accept"))
            {
                    setpen(146,208,80,1); // Green-ish
            }
            if(HasTag("TransactionStep_sort", "Execute"))
            {
                    setpen(76,76,76,1); // Gray-ish
            }
            // Standard flow
            if(HasTag("TransactionStep_sort", "Decline"))
            {
                    setpen(255,192,80,1); // Yellow-ish
            }
            if(HasTag("TransactionStep_sort", "Quit"))
            {
                    setpen(255,192,80,1); // Yellow-ish
            }
            if(HasTag("TransactionStep_sort", "Reject"))
            {
                    setpen(255,192,80,1); // Yellow-ish
            }
```

```
if(HasTag("TransactionStep_sort", "Stop"))
{
	setpen(255,192,80,1); // Yellow-ish
}
// Error flow
if(HasTag("TransactionStep_sort", "RevokeRequest"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokeAccept"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokePromise"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokeDeclare"))
{
	setpen(192,0,0,1); // Red-ish
}
// Error flow refuse
if(HasTag("TransactionStep_sort", "RevokeRequestRefuse"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokeAcceptRefuse"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokePromiseRefuse"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokeDeclareRefuse"))
{
	setpen(192,0,0,1); // Red-ish
}

// Error flow allow
if(HasTag("TransactionStep_sort", "RevokeRequestAllow"))
{
	setpen(192,0,0,1); // Red-ish
}
if(HasTag("TransactionStep_sort", "RevokeAcceptAllow"))
{
	setpen(192,0,0,1); // Red-ish
```

```
        }
        if(HasTag("TransactionStep_sort", "RevokePromiseAllow"))
        {
                setpen(192,0,0,1); // Red-ish
        }
        if(HasTag("TransactionStep_sort", "RevokeDeclareAllow"))
        {
                setpen(192,0,0,1); // Red-ish
        }
        rectangle(0,0,100,100);
}

shape TPDStep
{
        setfillcolor(255, 255, 255);
        setpen(0,0,0,1);
        // Happy flow
        if(HasTag("TransactionStep_sort", "Request"))
        {
                setpen(146,208,80,1); // Green-ish
        rectangle(0,0,100,100);
        ellipse(0,0,100,100);
        }
        if(HasTag("TransactionStep_sort", "Promise"))
        {
                setpen(146,208,80,1); // Green-ish
        rectangle(0,0,100,100);
        ellipse(0,0,100,100);
        }
        if(HasTag("TransactionStep_sort", "Declare"))
        {
                setpen(146,208,80,1); // Green-ish
        rectangle(0,0,100,100);
        ellipse(0,0,100,100);
        }
        if(HasTag("TransactionStep_sort", "Accept"))
        {
                setpen(146,208,80,1); // Green-ish
        rectangle(0,0,100,100);
        ellipse(0,0,100,100);
        }
        if(HasTag("TransactionStep_sort", "Execute"))
        {
                setpen(76,76,76,1); // Gray-ish
        rectangle(0,0,100,100);
            polygon(50, 50, 4, 40, 0);
}
```

```
// Standard flow
if(HasTag("TransactionStep_sort", "Decline"))
{
       setpen(255,192,80,1); // Yellow-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "Quit"))
{
       setpen(255,192,80,1); // Yellow-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "Reject"))
{
       setpen(255,192,80,1); // Yellow-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "Stop"))
{
       setpen(255,192,80,1); // Yellow-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
// Error flow
if(HasTag("TransactionStep_sort", "RevokeRequest"))
{
       setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokeAccept"))
{
       setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokePromise"))
{
       setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokeDeclare"))
{
       setpen(192,0,0,1); // Red-ish
```

```
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
// Error flow refuse
if(HasTag("TransactionStep_sort", "RevokeRequestRefuse"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokeAcceptRefuse"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokePromiseRefuse"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokeDeclareRefuse"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}

// Error flow allow
if(HasTag("TransactionStep_sort", "RevokeRequestAllow"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokeAcceptAllow"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
}
if(HasTag("TransactionStep_sort", "RevokePromiseAllow"))
{
	setpen(192,0,0,1); // Red-ish
rectangle(0,0,100,100);
ellipse(0,0,100,100);
```

```
            }
            if(HasTag("TransactionStep_sort", "RevokeDeclareAllow"))
            {
                   setpen(192,0,0,1); // Red-ish
            rectangle(0,0,100,100);
            ellipse(0,0,100,100);
            }

            // Discussion states
            if(HasTag("TransactionStep_sort", "Decline"))
            {
                   ellipse(10,10,90,90);
            }
            if(HasTag("TransactionStep_sort", "Reject"))
            {
                   ellipse(10,10,90,90);
            }
            if(HasTag("TransactionStep_sort", "RevokeRequest"))
            {
                   ellipse(10,10,90,90);
            }
            if(HasTag("TransactionStep_sort", "RevokeAccept"))
            {
                   ellipse(10,10,90,90);
            }
            if(HasTag("TransactionStep_sort", "RevokePromise"))
            {
                   ellipse(10,10,90,90);
            }
            if(HasTag("TransactionStep_sort", "RevokeDeclare"))
            {
                   ellipse(10,10,90,90);
            }

            if(HasTag("WillBeImplemented", "false"))
            {
                   setpen(255,0,0,3); // Red
                   moveto(0, 0);
        lineto(100, 100);
                   moveto(0, 100);
        lineto(100, 0);
            }
      }
}

shape label{
   setorigin("s",0,0);
```

```
    print("#alias#");
}
```

Listing G.6: Transaction Step shapescript

```
// ShapeScript StereoType: DEMO_ENTITY_TYPE

shape main{
    noshadow=true;

        //rounded rectangle
        if (HasTag("Composite␣Entity", "true")){
                setfillcolor(191,191,191);
                startpath();
                moveto(0,0);
                lineto(100,0);
                lineto(100,100);
                lineto(0,100);
                lineto(0,0);
                endpath();
                fillpath();
        } else {
        setfillcolor(255,255,255);
        }
    DrawNativeShape();
        setpencolor(255,255,255);
        setfillcolor(255,255,255);
        //triangle(0,0,10,10);
        startpath();
        moveto(0,0);
        lineto(10,0);
        lineto(0,10);
        lineto(0,0);
        endpath();
        fillpath();
        //triangle(90,0,100,10);
        startpath();
        moveto(90,0);
        lineto(100,0);
        lineto(100,10);
        lineto(90,0);
        endpath();
        fillpath();
        //triangle(90,90,100,100);
        startpath();
        moveto(90,100);
        lineto(100,100);
```

```
    lineto(100,90);
    lineto(90,100);
    endpath();
    fillpath();
    //triangle(0,90,10,100);
    startpath();
    moveto(0,90);
    lineto(10,100);
    lineto(0,100);
    lineto(0,90);
    endpath();
    fillpath();
    //rectangle
    setpen(255,255,255,4);
    moveto(0,0);
    lineto(100,0);
    lineto(100,100);
    lineto(0,100);
    lineto(0,0);

//rounded rectangle
    if (HasTag("Implementation␣Entity", "true")){
    setpen(0,255,255,2);
    } else {
    setpen(0,0,0,1);
    }
    moveto(50,0);
    lineto(10,0);
    bezierto(10,0,0,0,0,10);
    lineto(0,90);
    bezierto(0,90,0,100,10,100);
    lineto(90,100);
    bezierto(90,100,100,100,100,90);
    lineto(100,10);
    bezierto(100,10,100,0,90,0);
    lineto(50,0);

    shape dummy {
    }

    shape namesubshape {
            h_align = "center";
            v_align = "center";
            editablefield="name";

            println("#name#");
    }
```

```
}

shape label{
        editablefield = "alias";
        setorigin("s",0,0);
        print("#alias#");
}

shape ChildElement
{
        if(HasProperty("stereotype", "DEMO_ATTRIBUTE_TYPE"))
        {
              SetCompartmentName("Attributes");
        }

         AppendCompartmentText("#NAME#");
}
```

Listing G.7: Entity shapescript

```
shape main{
        roundrect(0,0,100,100,30,30);
        addsubshape("dummy",100,40);
        addsubshape("namesubshape",100,20);

        shape dummy {
        }

        shape namesubshape {
               h_align = "center";
               v_align = "center";
               editablefield="name";

               println("#name#");
        }

}
```

Listing G.8: Derived Entity type shapescript

```
shape main
{
        noShadow="true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
```

```
        setfixedregion(45,-5,55,5);
    moveto(45,-5);
    lineto(50,0);
    lineto(45,5);
}

shape source {
}

shape target {
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel {
// print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel {
   print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.9: Property Type shapescript

```
// ShapeScript StereoType: DEMO_actionrule_containment

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
```

```
    setfillcolor(getuserbordercolor());
    ellipse(-4,-4,4,4);
}

shape target
{
    setfillcolor(getuserbordercolor());
    startpath();
    moveto(0,0);
    lineto(10,4);
    lineto(10,-4);
    lineto(0,0);
    endpath();
    fillandstrokepath();
}


//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.10: Action Rule Containment shapescript

```
// ShapeScript StereoType: DEMO_ACTION_RULE_when

shape main{
    editablefield = "name";
    h_align = "center";
    v_align = "top";
    print("#alias#");

    moveto(0, 0);
```

```
    lineto(100, 0);
    lineto(100, 80);
    lineto(50, 100);
    lineto(0, 80);
    lineto(0, 0);
}
```

Listing G.11: Action Rule When shapescript

```
// ShapeScript StereoType: DEMO_ACTION_RULE_while

shape main{
    editablefield = "name";
    h_align = "center";
    v_align = "top";
    print("#alias#");

    setlinestyle("dash");
    polygon(50,50,8,50,22.5);

}
```

Listing G.12: Action Rule While shapescript

```
// ShapeScript StereoType: DEMO_ACTION_RULE_assess

shape main{
    editablefield = "name";
    h_align = "center";
    v_align = "top";
    print("#alias#");
    rectangle(0, 0, 100, 100);
}

shape ChildElement{
    editablefield = "name";
    h_align = "center";
    v_align = "top";
    print("#alias#");
    rectangle(0, 0, 100, 100);
}
```

Listing G.13: Action Rule Assess shapescript

```
// ShapeScript StereoType: DEMO_ACTION_RULE_then

shape main{
    editablefield = "name";
```

```
    h_align = "center";
    v_align = "top";
    print("#alias#");

    moveto(20, 10);
    lineto(60, 10);
    lineto(60, 0);
    lineto(100, 0);
    lineto(100, 100);
    lineto(60, 100);
    lineto(60, 90);
    lineto(20, 90);
    lineto(0, 50);
    lineto(20, 10);
}
```

Listing G.14: Action Rule Then shapescript

```
// ShapeScript StereoType: DEMO_ACTION_RULE_else

shape main{
    editablefield = "name";
    h_align = "center";
    v_align = "top";
    print("#alias#");

    moveto(20, 10);
    lineto(60, 10);
    lineto(60, 0);
    lineto(100, 0);
    lineto(100, 90);
    lineto(20, 90);
    lineto(0, 50);
    lineto(20, 10);
}
```

Listing G.15: Action Rule Else shapescript

G.5.2 Extra Elements

```
// ShapeScript StereoType: DEMO_WORKINSTRUCTION

shape main{
    h_align = "center";
    v_align = "top";
    editablefield = "name";
    DefSize(50,20);
```

```
    rectangle(0,0,100,100);
    rectangle(0,20,100,100);
    rectangle(0,40,100,100);
    rectangle(0,60,100,100);
    rectangle(0,80,100,100);
    print("#name#");
}

shape label
{
    editablefield="alias";
    setorigin("s",0,0);
    print("#alias#");
}
```

Listing G.16: Workinstruction shapescript

```
// ShapeScript StereoType: DEMO_DUPLICATE

shape main{
              DefSize(50,50);
       if (HasTag("ElementType", "TransactionKind"))
       {

              if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OCD"))
              {
                     AddSubShape("OCDTransactionKind", 100, 100, 0, 0);
              }
              else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
              {
                     if (hasproperty("diagram.stereotype", "DEMO4n")) {
                            AddSubShape("OCDTransactionKind", 100, 100, 0, 0);
                     }
                     else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                            AddSubShape("OCDTransactionKind", 100, 100, 0, 0);
                     }
                     else {
                            AddSubShape("PSDTransactionKind", 100, 100, 0, 0);
                     }
              }
              else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OFD"))
              {
                     AddSubShape("OFDTransactionKind", 100, 100, 0, 0);
              }
              else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_TPD"))
              {
```

```
        AddSubShape("TPDTransactionKind", 100, 100, 0, 0);
}
else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_SRD"))
{
        AddSubShape("SRDTransactionKind", 100, 100, 0, 0);
}
else {
 print("#diagram.mdgtype#");
}

shape OFDTransactionKind
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
        fixedAspectRatio = "true";
        DefSize(50,50);
        setpenwidth(2);
        //SetFillColor(255, 255, 255);
        //roundrect(0, 0, 100, 100, 30, 30);
        //SetPenColor(0, 0, 0);
        if (HasTag("transaction␣sort", "original"))
        {
                SetPenColor(255, 0, 0);
        }
        else if (HasTag("transaction␣sort", "informational"))
        {
                SetPenColor(0, 255, 0);
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
                SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
                SetPenColor(128, 128, 128);
        }
        setlinestyle("dot");
        polygon(50, 50, 4, 50, 0);
        print("#tag:Product␣Kind#");
}

shape SRDTransactionKind
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
```

```
        fixedAspectRatio = "true";
        DefSize(50,50);
        setlinestyle("dot");
        SetFillColor(127, 127, 127);

        SetPenColor(0, 0, 0);
        polygon(50, 50, 4, 71, 45);
        SetFillColor(191, 191, 191);
        polygon(50, 50, 4, 50, 0);
        print("#tag:Product␣Kind#");
}

shape OCDTransactionKind
{
        h_align = "center";
        v_align = "center";
        editablefield = "name";
        fixedAspectRatio = "true";
        DefSize(50,50);
        setpenwidth(2);
        SetFillColor(255, 255, 255);

        setlinestyle("dot");
        if (HasTag("ECF␣Missing", "true"))
        {
                setlinestyle("dashdot");
        }
        SetPenColor(0, 0, 0);
        Ellipse(0, 0, 100, 100);
        setlinestyle("solid");
        setlinestyle("dot");

        if (HasTag("transaction␣sort", "original"))
        {
                SetPenColor(255, 0, 0);
        }
        else if (HasTag("transaction␣sort", "informational"))
        {
                SetPenColor(0, 255, 0);
        }
        else if (HasTag("transaction␣sort", "documental"))
        {
                SetPenColor(0, 0, 255);
        }
        else if (HasTag("transaction␣sort", "fysical"))
        {
                SetPenColor(128, 128, 128);
```

```
            }
            polygon(50, 50, 4, 50, 0);
            print("#name#");
    }

    shape PSDTransactionKind
    {
            defsize(200, 40);
        setpenwidth(2);
        setlinestyle("dot");
        roundrect(0, 0, 100, 100, 100, 100);
            fillandstrokepath();
            addsubshape("ex", 20, 100, 40, 0);
    }

    shape TPDTransactionKind
    {
            defsize(50, 50);
        setpenwidth(2);
        setlinestyle("dot");
        roundrect(0, 0, 100, 100, 100, 100);
            fillandstrokepath();
            addsubshape("ex", 20, 100, 40, 0);
    }

    shape ex
    {
h_align = "center";
v_align = "center";
            editablefield = "name";
            setlinestyle("dot");
            setpenwidth(2);
println("#name#");
if (HasTag("transaction␣sort", "original"))
{
    SetPenColor(255, 0, 0);
}
else if (HasTag("transaction␣sort", "informational"))
{
    SetPenColor(0, 255, 0);
}
else if (HasTag("transaction␣sort", "documental"))
{
    SetPenColor(0, 0, 255);
}
else if (HasTag("transaction␣sort", "fysical"))
{
```

```
            SetPenColor(128, 128, 128);
        }
        polygon(50, 50, 4, 50, 0);
}

    }
    else
    if (HasTag("ElementType", "ElementaryActorRole"))
    {
            if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
                    if (hasproperty("diagram.stereotype", "DEMO4n")) {
                            AddSubShape("PSDNormal", 100, 100, 0, 0);
                    }
                    else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                            AddSubShape("PSDNormal", 100, 100, 0, 0);
                    }
                    else {
                            AddSubShape("PSDSwimlane", 100, 100, 0, 0);
                    }
            }
            else {
                    AddSubShape("PSDNormal", 100, 100, 0, 0);
            }

            shape PSDSwimlane
            {
                    h_align = "left";
                    v_align = "top";
                    editablefield = "name";
                    DefSize(30,200);
                    setpen(220,220,220,3);
                    moveto(0,100);
                    lineto(100,100);
                    print("#name#");
            }
            shape PSDNormal
            {
                    h_align = "center";
                    v_align = "top";
                    editablefield = "name";
                    DefSize(50,50);
                    // ECF Missing // ECF 1, 10, 20
                    setpenwidth(2);
                    setlinestyle("dot");
                    if (HasTag("ECF␣Missing", "true"))
               {
                        setlinestyle("dashdot");
```

```
            }
            rectangle(0,0,100,100);
            print("#name#");
        }
}
else
if (HasTag("ElementType", "CompositeActorRole"))
{
        if (haspropertу("diagram.mdgtype", "DEMO::DEMO_PSD")) {
                if (haspropertу("diagram.stereotype", "DEMO4n")) {
                        AddSubShape("CARNormal", 100, 100, 0, 0);
                }
                else if (haspropertу("diagram.stereotype", "DEMO4c")) {
                        AddSubShape("CARNormal", 100, 100, 0, 0);
                }
                else {
                        AddSubShape("PSDSwimlane", 100, 100, 0, 0);
                }
        }
        else {
                AddSubShape("CARNormal", 100, 100, 0, 0);
        }

        shape PSDSwimlane
        {
                h_align = "left";
                v_align = "top";
                editablefield = "name";
                DefSize(30,200);
                if (HasProperty('rectanglenotation', '0'))
                {
                }
                else
                {
                }
                moveto(0,100);
                lineto(100,100);
                print("#name#");
        }
                shape CARNormal
        {
                h_align = "center";
                v_align = "top";
                editablefield = "name";
                DefSize(50,50);
                setpenwidth(2);
                setlinestyle("dot");
```

```
                // ECF Missing // ECF 1, 10, 20
                if (HasTag("ECF␣Missing", "true"))
                {
                        setlinestyle("dashdot");
                }
                if (HasProperty('rectanglenotation', '0'))
                {
                        setfillcolor(220, 220, 220);
                        rectangle(0, 0, 100, 100);
                        // code for iconic representation
                }
                else
                {
                        setpen(220, 220, 220, 5);
                        rectangle(0, 0, 100, 100);
                        // code for rectangular representation
                        // e.g. Rectangle or DrawNativeShape
                }
                rectangle(0,0,100,100);
                print("#name#");
        }
}
else
if (HasTag("ElementType", "AggregateTransactionKind"))
{
        DefSize(50,50);
        setfillcolor(220,220,220);
        setlinestyle("dot");
        setpenwidth(2);

        ellipse(5,0,105,100);
        ellipse(0,0,100,100);
        if (hasproperty("diagram.stereotype", "DEMO4n")) {
                if (HasTag("transaction␣sort", "original"))
                {
                        SetPenColor(255, 0, 0);
                }
                else if (HasTag("transaction␣sort", "informational"))
                {
                        SetPenColor(0, 255, 0);
                }
                else if (HasTag("transaction␣sort", "documental"))
                {
                        SetPenColor(0, 0, 255);
                }
                else if (HasTag("transaction␣sort", "fysical"))
                {
```

```
                        SetPenColor(128, 128, 128);
                }
        }
        else if (hasproperty("diagram.stereotype", "DEMO4c")) {
                if (HasTag("transaction␣sort", "original"))
                {
                        SetPenColor(255, 0, 0);
                }
                else if (HasTag("transaction␣sort", "informational"))
                {
                        SetPenColor(0, 255, 0);
                }
                else if (HasTag("transaction␣sort", "documental"))
                {
                        SetPenColor(0, 0, 255);
                }
                else if (HasTag("transaction␣sort", "fysical"))
                {
                        SetPenColor(128, 128, 128);
                }
        }
        polygon(50,50,4,50,0);

        print("#name#");
    }
    else
    {
        // question mark in 3 sections
        setpen(255, 0, 0, 2);
        moveto(50, 90);
        lineto(50, 70);
        arcto(20, 10, 80, 70, 50, 70, 80, 40);
        arcto(20, 10, 80, 70, 80, 40, 20, 40);
    }
    shape PSDSwimlane
    {
        h_align = "left";
        v_align = "top";
        editablefield = "name";
        DefSize(30,200);
        setpen(220,220,220,3);
        setlinestyle("dot");
        moveto(0,100);
        lineto(100,100);
        print("#name#");
    }
}
```

```
shape label{
	editablefield = "alias";
	if (HasTag("ElementType", "TransactionKind"))
	{
		setorigin("s",0,0);
		if (hasproperty("diagram.stereotype", "DEMO4n")) {
		}
		else if (hasproperty("diagram.stereotype", "DEMO4c")) {
		}
		else {
			print("#alias#");
		}
	}
	else
	{
		if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
			if (hasproperty("diagram.stereotype", "DEMO4n")) {
				setorigin("center", 0, 0);
			}
			else if (hasproperty("diagram.stereotype", "DEMO4c")) {
				setorigin("center", 0, 0);
			}
			else
			{
				setorigin("w", 0, 0);
			}
		}
		else {
			setorigin("s", 0, 0);
		}
		//print("#alias#");
		if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OCD"))
		{
			print("#alias#");
		}
		else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
		{
			if (hasproperty("diagram.stereotype", "DEMO4n")) {
				print("#alias#");
			}
			else if (hasproperty("diagram.stereotype", "DEMO4c")) {
				print("#alias#");
			}
			else {
				print("#alias#");
			}
```

```
                }
                else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OFD"))
                {
                        print("#tag:Product␣Kind␣Formulation#");
                }
                else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_TPD"))
                {
                        print("#alias#");
                }
                else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_SRD"))
                {
                        print("#alias#");
                }
                else {
                        print("#diagram.mdgtype#");
                }
        }
}
```

Listing G.17: Duplicate shapescript

```
// ShapeScript StereoType: DEMO_SECURITY_FUNCTION

shape main{
    h_align = "center";
    v_align = "top";
    editablefield = "name";
        defsize(50,50);
        setpencolor(255,0,255);
        rectangle(0,0,100,100);

        print("#name#");
}

decoration lock {
        orientation = "NE";

        rectangle(20,40,80,100);
        setfillcolor(255,0,255);
        fillpath();
        setfillcolor(255,255,255);
        rectangle(45,60,55,80);
        fillpath();

        setpencolor(255,0,255);
        moveto(70,40);
        setpen(255,0,255,2);
```

```
        arcto(30,10,70,60,70,40,50,0);
        arcto(30,10,70,60,50,00,30,40);
}

shape label
{
    editablefield="alias";
    setorigin("s",0,0);
    print("#alias#");
}
```

Listing G.18: Security Function shapescript

```
// ShapeScript StereoType: DEMO_SECURITY_ROLE

shape main{
    h_align = "center";
    v_align = "top";
    editablefield = "name";
        defsize(50,50);
        setpencolor(255,0,96);
        rectangle(0,0,100,100);

        print("#name#");
}

decoration lock {
        orientation = "NE";

        setpencolor(255,0,96);
        setfillcolor(255,0,96);
        ellipse(20,40,80,100);
        //setfillcolor(255,0,127);
        fillpath();
        setfillcolor(255,255,255);
        rectangle(45,60,55,80);
        fillpath();

        setpencolor(255,0,96);
        moveto(70,40);
        setpen(255,0,96,2);
        arcto(30,10,70,60,70,50,50,0);
        arcto(30,10,70,60,50,00,30,50);
}

shape label
{
```

```
    editablefield="alias";
    setorigin("s",0,0);
    print("#alias#");
}
```

Listing G.19: Security Role shapescript

```
// ShapeScript StereoType: DEMO_ACTOR

shape main{
    h_align = "center";
    v_align = "top";
    editablefield = "name";
    DefSize(50,50);

    if (HasTag("internal", "true")){
                setfillcolor(255,255,192);
        } else {
                setfillcolor(200,200,32);
        }
    rectangle(0,0,100,100);
    print("#name#");
}

shape label{
    editablefield = "alias";
    setorigin("s",0,0);
    print("#alias#");
}
```

Listing G.20: Actor shapescript

```
// ShapeScript StereoType: DEMO_SoI_BOUNDARY

shape main{
    editablefield = "name";
    h_align = "right";
    v_align = "top";
    print("#name#");

    DefSize(50, 50);
    setfillcolor(255, 255, 255);
    setpen(128,128,128,6);
    rectangle(0,0,100,100);
}

shape label{
```

```
}
```

Listing G.21: Boundary shapescript

```
shape main{
    h_align = "center";
    v_align = "center";
    editablefield = "name";
        defsize(50,50);
        ellipse(0,0,100,100);
        moveto(50,0);
        lineto(50,100);
        moveto(0,50);
        lineto(100,50);

        setpen(0,127,0,6);
        //Knowledge
        moveto(28,23);
        lineto(28,45);
        moveto(39,23);
        lineto(30,34);
        lineto(39,45);

        //Abilities
        moveto(61,45);
        lineto(67,23);
        lineto(73,45);
        moveto(63,38);
        lineto(71,38);

        //Motivation


        //Personality


   print("#name#");
}

shape label
{
    editablefield="alias";
    setorigin("s",0,0);
    print("#alias#");
}
```

Listing G.22: Competence shapescript

```
shape main{
       rectangle(0,0,100,100);
       print("#alias#");
}
```

Listing G.23: Interview Line shapescript

```
shape main{
   noshadow=true;
   DrawNativeShape();
       }
```

Listing G.24: Conversion Rule shapescript

G.5.3 Connectors

```
// ShapeScript StereoType: DEMO_initiator

shape main
{
   noShadow = "true";

   SetLineStyle("solid");
   if (hasproperty("diagram.stereotype", "DEMO4c")) {
   }
   else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
   }
   else
   {
       moveto(0, 0);
       lineto(100, 0);
   }
}

shape source
{
   // No source side drawing
}

shape target
{
   // No target side drawing
}

//shape LeftTopLabel {
// print("LeftTopLabel");
```

```
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.25: Initiator shapescript

```
// ShapeScript StereoType: DEMO_executor

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    if (hasproperty("diagram.stereotype", "DEMO4n")) {
    }
    else if (hasproperty("diagram.stereotype", "DEMO4c")) {
    }
    else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
        }
    else
    {
        moveto(0,0);
        lineto(100,0);
    }
}

shape source
{
    // No source side drawing
}

shape target
{
    // Target side an black filled diamond
```

```
    if (hasproperty("diagram.stereotype", "DEMO4n")) {
    }
    else if (hasproperty("diagram.stereotype", "DEMO4c")) {
    }
    else if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD")) {
        }
        else
    {
        SetFillColor(0,0,0);
        polygon(0,0,4,5,0);
    }
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.26: Executor shapescript

```
// ShapeScript StereoType: DEMO_bank_access

shape main
{
    noShadow = "true";

    SetLineStyle("dash");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
```

```
    // No source side drawing
}

shape target
{
    // No target side drawing
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.27: Bank Access shapescript

```
// ShapeScript StereoType: DEMO_Containedin

shape main
{
    noShadow = "true";

    SetLineStyle("dash");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       StartPath();
```

```
        MoveTo(0,0);
        LineTo(15,9);
        LineTo(15,-9);
        EndPath();
        FillAndStrokePath();
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
        // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
        print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.28: ContainedIn shapescript

```
// ShapeScript StereoType: DEMO_call_link

shape main{
    noshadow = true;
    SetLineStyle("solid");
        setpencolor(255,255,0);
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
                setpencolor(0,0,0);
    }
        // draw a solid line
        MoveTo(0,0);
        LineTo(100,0);
}

shape source{
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
        Ellipse(-5,-5,5,5);
```

```
    }
}

shape target{
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
        MoveTo(0,0);

        SetFillColor(0,0,0);
        polygon(10,0,3,5,180);

        SetFillColor(252,242,227); // default color
        Rectangle(-5,-5,5,5);
    }
}

//shape LeftTopLabel {
// print("#tag:From␣C-fact#");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    // print("MiddleBottomLabel");
}
// shape RightBottomLabel {
// print("#tag:To␣C-act#");
// }
```

Listing G.29: Call shapescript

```
// ShapeScript StereoType: DEMO_wait_link

shape main{
    // draw a dashed line
    noshadow = true;
        SetLineStyle("dash");
    setpencolor(255,255,0);
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
                setpencolor(0,0,0);
```

```
    }
        MoveTo(0, 0);
        LineTo(100, 0);
}

shape source{
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
        Ellipse(-5,-5,5,5);
    }
}

shape target{
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_PSD"))
    {
        MoveTo(0,0);

        SetFillColor(0,0,0);
        polygon(10,0,3,5,180);

        SetFillColor(252,242,227); // default color
        Rectangle(-5,-5,5,5);
    }
}

//shape LeftTopLabel {
// print("#tag:From␣C-fact#");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    // print("MiddleBottomLabel");
}
// shape RightBottomLabel {
// print("#tag:To␣C-act#");
// }
```

Listing G.30: Wait shapescript

```
// ShapeScript StereoType: DEMO_steprule
```

```
shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
}

shape target
{
    setfillcolor(getuserbordercolor());
    startpath();
    moveto(0,0);
    lineto(10,4);
    lineto(10,-4);
    lineto(0,0);
    endpath();
    fillandstrokepath();
}


//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.31: Step rule shapescript

```
// ShapeScript StereoType: DEMO_reference

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);

    setfixedregion(40,-10,60,10);
    SetLineStyle("solid");

    moveto(45,-5);
    lineto(55,0);

    moveto(55,0);
    lineto(45,5);
}

shape source{
}

shape target{
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.32: Reference shapescript

```
// ShapeScript StereoType: DEMO_aggregation

shape main{ // draw a dashed line
    noshadow = true;
    SetLineStyle("dash");
    MoveTo(0,0);
    LineTo(100,0);
}

shape target{ // draw an arrowhead at the target end
    StartPath();
    MoveTo(0,0);
    LineTo(30,18);
    LineTo(30,-18);
    EndPath();
    FillAndStrokePath();

    MoveTo(15, 0);
    LineTo(27, 0);
    MoveTo(21, 6);
    LineTo(21, -6);

    MoveTo(17, 4);
    LineTo(25, -4);
    MoveTo(25, 4);
    LineTo(17, -4);
}

//shape LeftTopLabel {
// print("#tag:From␣C-fact#");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    // print("MiddleBottomLabel");
}
// shape RightBottomLabel {
// print("#tag:To␣C-act#");
// }
```

Listing G.33: Aggregation shapescript

```
// ShapeScript StereoType: DEMO_specialisation

shape main{ // draw a dashed line
    noshadow = true;
    SetLineStyle("dash");
    MoveTo(0,0);
    LineTo(100,0);
}

shape target{ // draw an arrowhead at the target end
        setfillcolor(255,255,255);
    StartPath();
    MoveTo(0,0);
    LineTo(10,5);
    LineTo(10,-5);
    LineTo(0,0);
    EndPath();
    FillAndStrokePath();
}
```

Listing G.34: Specialisation shapescript

```
// ShapeScript StereoType: DEMO_generalisation

shape main{ // draw a dashed line
        noshadow = true;
        SetLineStyle("dash");
        MoveTo(0,0);
        LineTo(100,0);
}

shape target{ // draw an arrowhead at the target end
        StartPath();
        MoveTo(0,0);
        LineTo(30,18);
        LineTo(30,-18);
        EndPath();
        FillAndStrokePath();

        MoveTo(15,0);
        LineTo(27,0);
        MoveTo(21,6);
        LineTo(21,-6);
}
```

```
//shape LeftTopLabel {
// print("#tag:From␣C-fact#");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       // print("MiddleBottomLabel");
}
// shape RightBottomLabel {
// print("#tag:To␣C-act#");
// }
```

Listing G.35: Generalisation shapescript

```
// ShapeScript StereoType: DEMO_excludes

shape main
{
    noShadow = "true";

    SetLineStyle("dash");
    moveto(0,0);
    lineto(100,0);

    setfixedregion(40,-10,60,10);
    SetLineStyle("solid");
    Ellipse(40,-10,60,10);
    moveto(45,-5);
    lineto(55,5);

    moveto(55,-5);
    lineto(45,5);
}

shape source{
}

shape target{
}
```

```
//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.36: Exclude shapescript

```
// ShapeScript StereoType: DEMO_precedence

shape main
{
    noShadow = "true";

    SetLineStyle("dash");
    moveto(0,0);
    lineto(100,0);
}

shape source{
}

shape target{
    setfillcolor(getuserbordercolor());
    SetLineStyle("solid");

    moveto(10,-5);
    lineto(0,0);

    moveto(10,5);
    lineto(0,0);
}
```

```
//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.37: Precedence shapescript

```
// ShapeScript StereoType: DEMO_preclusion

shape main
{
    noShadow = "true";

    SetLineStyle("dash");
    moveto(0,0);
    lineto(100,0);

    setfixedregion(40,-10,60,10);
    SetLineStyle("solid");
    Ellipse(40,-10,60,10);
    moveto(45,-5);
    lineto(55,5);

    moveto(55,-5);
    lineto(45,5);
}

shape source{
}

shape target{
    setfillcolor(getuserbordercolor());
    SetLineStyle("solid");
```

```
    moveto(0,-5);
    lineto(10,0);

    moveto(0,5);
    lineto(10,0);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.38: Preclusion shapescript

```
shape main { // draw a dashed line
  noshadow=true;
  SetLineStyle("solid");
  MoveTo(0,0);
  LineTo(100,0);
}

shape source {
}

shape target {
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel {
// print("MiddleTopLabel");
```

```
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel {
   print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.39: Concerns shapescript

```
// ShapeScript StereoType: DEMO_withclause

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
}

shape target
{
    setfillcolor(getuserbordercolor());
    startpath();
    moveto(0,0);
    lineto(10,4);
    lineto(10,-4);
    lineto(0,0);
    endpath();
    fillandstrokepath();
}


//shape LeftTopLabel {
// print("LeftTopLabel");
// }
```

```
shape MiddleTopLabel{
    // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
    print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.40: With clause shapescript

```
// ShapeScript StereoType: DEMO_hidden

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target
{
    // No target side drawing
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
        // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
```

```
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.41: Link CM Hidden shapescript

G.5.4 Extra Connectors

```
// ShapeScript StereoType: DEMO_describes

shape main{
    noshadow = true;
    if (hasproperty("diagram.mdgtype", "DEMO::DEMO_OCD"))
    {
        // draw a solid line
        SetLineStyle("solid");
        MoveTo(0,0);
        LineTo(100,0);
    } else
        {
               SetPencolor(255,0,0);
        // draw a solid line
        SetLineStyle("solid");
        MoveTo(0,0);
        LineTo(100,0);
        }
}
```

Listing G.42: Describes shapescript

```
// ShapeScript StereoType: DEMO_ManagerOf

shape main
{
    noShadow = "true";

    SetLineStyle("dot");
    moveto(0,0);
    lineto(100,0);
}
```

```
shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
        MoveTo(7,4);
        LineTo(0,0);
        LineTo(7,-4);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
        // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
        print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.43: Executed By shapescript

```
// ShapeScript StereoType: DEMO_ManagerOf

shape main
{
    noShadow = "true";

    SetLineStyle("dashdot");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
```

```
}

shape target{ // draw an arrowhead at the target end
       MoveTo(7,4);
       LineTo(0,0);
       LineTo(7,-4);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.44: Managed By shapescript

```
// ShapeScript StereoType: DEMO_hidden

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       MoveTo(15,9);
```

```
        LineTo(0,0);
        LineTo(15,-9);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
        // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
        print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.45: Measurement shapescript

```
// ShapeScript StereoType: DEMO_MeasuredBy

shape main
{
    noShadow = "true";

    SetLineStyle("dot");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
        MoveTo(7,4);
        LineTo(0,0);
        LineTo(7,-4);
}
```

```
//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.46: Measuring shapescript

```
// ShapeScript StereoType: DEMO_hidden

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       MoveTo(15,9);
       LineTo(0,0);
       LineTo(15,-9);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
```

```
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.47: Needs shapescript

```
// ShapeScript StereoType: DEMO_processorder

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
}

shape target
{
    setfillcolor(getuserbordercolor());
    startpath();
    moveto(0,0);
    lineto(10,4);
    lineto(10,-4);
    lineto(0,0);
    endpath();
    fillandstrokepath();
}


//shape LeftTopLabel {
// print("LeftTopLabel");
```

```
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.48: Process Order shapescript

```
// ShapeScript StereoType: DEMO_ManagerOf

shape main
{
    noShadow = "true";

    SetLineStyle("dashdotdot");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       MoveTo(7,4);
       LineTo(0,0);
       LineTo(7,-4);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
```

```
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.49: Role Of shapescript

```
// ShapeScript StereoType: DEMO_hidden

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       MoveTo(15,9);
       LineTo(0,0);
       LineTo(15,-9);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
```

```
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.50: Supporting shapescript

```
// ShapeScript StereoType: DEMO_MeasuredBy

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       setfillcolor(0,0,0);
       setpen(0,0,0);
       startpath();
       MoveTo(7,4);
       LineTo(0,0);
       LineTo(7,-4);
       endpath();
       fillandstrokepath();
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
```

```
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
       print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.51: Trigger shapescript

```
// ShapeScript StereoType: DEMO_hidden

shape main
{
    noShadow = "true";

    SetLineStyle("solid");
    moveto(0,0);
    lineto(100,0);
}

shape source
{
    // No source side drawing
}

shape target{ // draw an arrowhead at the target end
       MoveTo(15,9);
       LineTo(0,0);
       LineTo(15,-9);
}

//shape LeftTopLabel {
// print("LeftTopLabel");
// }
shape MiddleTopLabel{
       // print("MiddleTopLabel");
}
// shape RightTopLabel {
// print("RightTopLabel");
// }
// shape LeftBottomLabel {
// print("LeftBottomLabel");
// }
shape MiddleBottomLabel{
```

```
        print("#name#");
}
// shape RightBottomLabel {
// print("RightBottomLabel");
// }
```

Listing G.52: In interview shapescript