FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Conceptualisation: Chapters from Harmonising Enterprise and Software Engineering

by

Robert Pergl

A habilitation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague

Prague, September 2018

Department of Software Engineering
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Head of Department:
Ing. Michal Valenta, Ph.D.

Abstract

This habilitation thesis deals with leveraging conceptualisation for harmonising software engineering in the context of enterprise engineering. It is based on the research and work done in the Centre of Conceptual Modelling and Implementation (CCMi) at the Department of Software Engineering since its foundation by the author in 2012. Theoretical foundations and key methodologies are introduced. Past and ongoing research and applications based on them are then elaborated on; These consist of several layers: (i) Methodological Studies, (ii) Methodological Improvements, (iii) Tooling and (iv) Applications. Achieved results at the levels of bachelor and diploma theses, Ph.D. and post-doc research projects are discussed. Key international collaborations, both academic and from practice are also explained. The core of the thesis then consists of chapters being key already-published, peer-reviewed publications.

Keywords:

conceptual modelling, ontologies, UML, OntoUML, UFO, BPMN, enterprise engineering, business process management, DEMO, model-driven engineering, models transformation, evolvability, normalised systems, software engineering, models execution, workflow, CASE.

Acknowledgements

My deep gratitude belongs to my family, especially my wife Hanka, who devotedly supports me in my endeavours both mentally and practically. I wish all professors to have such a wonderful wife!

Starting from the beginnings, I would like to express a deep gratitude to a great evangelist *doc. Ing. Vojtěch Merunka, Ph.D.* for kindling the spark in me to pursue the academic path and opening many wonderful doors to exciting knowledge and opportunities.

I am grateful to late *prof. RNDr. Jiří Vaníček, CSc.* for opening the golden vault of theoretical foundations of computer science to me and being my kind mentor during uneasy explorations.

My sincere thanks go to my superior and dear friend, *Ing. Michal Valenta, Ph.D.*, an exemplar of an enlightened leader, whose support, wisdom, understanding and excitement give a unique soul to our family — the Department of Software Engineering. Similarly, my many thanks go to our first Dean, *prof. Ing. Pavel Tvrdík, CSc.*. The immense support and trust of these two outstanding personalities enabled founding of the Centre for Conceptual Modelling and Implementation and its successes. I am happy and grateful that I found the same trust and support in our current Dean, *doc. RNDr. Ing. Marcel Jiřina, Ph.D.*.

An immense merit goes then to the members of CCMi — my colleagues, PhD, master and bachelor students who have done a lot of hard work with me (or without me) on the research and projects; many thanks for their enthusiasm and making CCMi great! I would like to namely thank *prof. Dr. Ing. Petr Kroha, CSc.*, my kind mentor and *Ing. Zdeněk Rybola, Ph.D.*, who tirelessly pushed forward our research on OntoUML for software engineering. My thanks also go to our wonderful department assistant, *Mgr. Alena Libánská*, for all her help, smile and warm heart. And there are many more outstanding people at our Faculty who also have my admiration and thanks, the acknowledgements to whom could easily fill the whole text.

Now going abroad, my first thoughts go to *prof. Giancarlo Guizzardi*, an exceptional scientist and person, whose work has been literally the corner-stone of CCMi. Every meeting with Giancarlo is a great pleasure, every talk is life-changing and eyes-opening.

Going further in time, the same thanks for bringing an essential and foundational work

then goes to the "father of enterprise engineering", the esteemed *prof. Jan L.G. Dietz*, whose kindness and trust in our potential laid foundations in many of our successes and his invitation to the CIAO! Network opened a wonderful world of topics, collaborations and friendships.

My thanks goes to the wonderful CIAO! community consisting of excellent scientists and professionals, whose enthusiasm and knowledge changes the world. My special thanks go to *prof. Jan Verelst* for research inspiration and substantial support and belief in our potential that has brought several trails of intensive and fruitful collaboration. Next special thanks go to *Dr. Steven van Kervel*, the founder of ForMetis Consultants and a person with a golden human character, whose merit is DEMO embedding in practice; we are honoured he took us to sail that ship with him. A substantial funding of ForMetis to CCMi members will also not be forgotten.

And arriving to the "implementation" part, we owe a lot to *prof. Stéphane Ducasse*, a true visionary and the driving force of the Pharo community, who has been helping and supporting us tremendously.

Recently, we also found a nice open community in the body of the ELIXIR infrastructure. Here go many thanks to the Head of ELIXIR Czech Node, *RNDr. Jiří Vondrášek, Ph.D.* for accepting us warmly in the community, opening opportunities of contribution to us and giving us much support in the hard beginnings. Also, this would not happen without *Dr. Luiz Olavo Bonino*, who had persuaded me that we can bring a value to ELIXIR. My thanks also go to Netherlands to amazing *prof. Dr. Barend Mons*, a visionary now changing the data world to a better, FAIR future, who gave us the opportunity to start leveraging our skills for this great purpose. Here, I must not forget *Dr. Rob Hooft* — it is a real pleasure to cooperate with a genius mind.

Last but not least, I would like to acknowledge a place — a cottage built my grand parents near the village Žloukovice, where my thoughts flow as nicely as the river Berounka.

Dedication

In loving memory of my dear mother, Ing. Bohdanka Perglová (1946-2018). My deepest gratitude.

. . .

Contents

Chapter 1

Foreword of the Author

Software engineering is an ultimate human design endeavour. Its variance of forms and possibilities surpasses all other design disciplines. The technologies have gone so far that today's limits are rather human imagination, complexity management and requirements engineering.

This is why the importance of conceptual modelling is rising in software engineering, as well as in other disciplines today — as a human-centred knowledge base of the domain, a vehicle of shared reasoning and complexity management; all this in a crisp-clear form, eradicating all mists of vagueness, misinterpretation and misunderstanding.

Thanks to technological advancements, software engineering and enterprise engineering became close engineering relatives — the younger brother software engineering being now the right hand of the traditional enterprise engineering discipline: information and knowledge systems became the neural network of human coordination, cooperation and co-production.

At the same time, technological, managerial and other advancements have not achieved a perfect harmony in the enterprise engineering family, yet. The success rate of software projects is rising, however, the level is still not satisfactory.

Christopher Alexander explains in his legendary work "Notes on the Synthesis of Form" [1]:

> The *form* is a part of the world over which we have control, and which we decide to shape while leaving the rest of the world as it is. The *context* is that part of the world which puts demands on this form; anything in the world that makes demands of the form is context. *Fitness* is a relation of mutual acceptability between these two. In a problem of design we want to satisfy the mutual demands which the two make on one another. We want to put the context and the form into *effortless contact or frictionless coexistence.*

In my endeavour, the context is enterprise engineering and the form is software engineering. This is why I speak about "harmonising enterprise and software engineering", as,

together with Alexander, I believe that speaking about "fitness" of software engineering can take place just in a right context.

Alas, the topic of finding a proper fit of software engineering in the context of enterprise engineering is naturally so vast that it will probably be never solved in full, as Alexander warns us:

> In general, unfortunately, we cannot give an adequate description of the context we are dealing with. The fields of the contexts we encounter in the real world cannot be described in the unitary fashion ... There is as yet no theory of ensembles capable of expressing a unitary description of the varied phenomena we encounter in the urban context of a dwelling, for example, or in a sonata, or a production cycle.

Nevertheless, 15 years of research and practice established a strong belief in me that *conceptual modelling* together with proper engineering methods, techniques and tools can push the state-of-the art significantly, while, at the same time not being leveraged enough in the main-stream software-engineering research outlets and practice — a doubting reader can themselves make a small survey of a grievous ratio of number of software engineering institutes and management institutes compared to conceptual modelling and ontology institutes.

Throughout the years, I tackled several research topics in the area. Apart from being a challenging and satisfying endeavour, the work has shown high practical relevance and opened more research topics than it actually solved. This formed into the idea of conceiving a dedicated research and practising group, which was founded in 2012 and named "Centre for Conceptual Modelling". The idea was highly inspired by the world leading conceptual modelling research group *Núcleo de Estudos em Modelagem Conceitual e Ontologias) (NEMO)*. As more implementation-related topics started to occur, we extended the name to "Centre for Conceptual Modelling and Implementation" [2].

The presented work brings now a collection of the work done mainly in the first 6 years of CCMi. Its main part consists of chapters addressing individual research problems in the field. They are already published and reviewed papers. The introductory part then explains the context, motivation and especially relations between the topics, which may seem various, but there is a strong silver thread binding them, or more precisely a cobweb of silver threads giving a coherent whole. Some of the parts present tangible results in various stages of maturity and adoption, some of them are more a manifesto of open research problems that we start to tackle.

The targeted readers of this work are fellow researchers interested in our work and, possibly, a collaboration. Students and new colleagues will find this publication helpful in acquainting themselves with the topics of their interest and future occupation, being it a bachelor's thesis, a master's thesis, a PhD thesis, or to just peek into the topics of one of the research groups at FIT CTU.

Robert Pergl

In Žloukovice, August 2018

Chapter 2

Structure of the Thesis

Part I contains an introduction to all research topics performed in CCMi. Most of them were conceived in years before CCMi, however the major work was done between years 2012-2018. The Introduction presents a brief overview of each topic and open research questions.

Figure 3.1 can be seen as a conceptual map of the text that begins from the centre by introducing "the heart" — conceptualisation — in Chapter 3, then continuing to the left to enterprise engineering in Chapter 4, then going right by describing the subtle relation between conceptual modelling and implementation in Chapter 5, emerging into the topic of software implementation in Chapter 7 and tooling in Chapter 8. Chapter 6 about evolvability and Normalised Systems then permeates the whole map in Figure 3.1, so it, kind of, goes into the third dimension.

The key sections along the way are "Our Contribution and Applications", where our results, ongoing and planned research are presented.

Key contributions authored and co-authored by the author then form the chapters of Part II. As it is not possible to order them by a single key, they are present in the order of appearance in the Introduction and Table 2.1 summarises the topics involved. A diagram on Figure 2.1 then presents another, level-based, view on the work done.

Chapters presented in their original form of camera-ready published papers have the following advantages, why the author decided to persist them:

- All the papers were peer-reviewed; presenting them in their original form does not introduce new editorial bugs.
- The reader does not need to search the original publications to make precise citations.
- Each chapter contains its own list of references and sections numbering, references are local, which makes it more convenient in a long and heterogeneous work.
- Each chapter is "self contained" with its own introduction and context, so the reader may conveniently jump directly to the topic of interest without loosing a context;

On the other hand, some information is duplicated, however, the reader may skip an introduction easily.

Table 2.1: Overview of the chapters

Chapter		Notations and Methodologies					Topics				
		UML	Onto-UML	BPMN	BORM	DEMO	Ontology	MDE	Implementation	Simulation and Validation	Tooling
10	Supporting Enterprise IS Modelling using Ontological Analysis				X		X	X			
11	Instance-Level Modelling and Simulation Revisited	X	X				X			X	
12	Towards OntoUML for Software Engineering: From Domain Ontology to Implementation Model	X	X				X	X			
13	Towards Formal Foundations for BORM ORD Validation and Simulation				X		X			X	
14	The Prefix Machine – a Formal Foundation for the BORM OR Diagrams Validation and Simulation				X		X			X	
15	Revisiting the BORM OR Diagram Composition Pattern				X		X				
16	Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment			X		X	X				
17	Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method			X		X	X	X			
18	Modelling and Prototyping of Business Applications Based on Multilevel Domain-Specific Language						X		X		
19	Analysing Functional Paradigm Concepts - The JavaScript Case						X		X		
20	OpenCASE - A Tool for Ontology-Centred Conceptual Modelling				X					X	X
21	BORM-II and UML as Accessibility Process in Knowledge and Business Modelling				X		X			X	X
22	The OpenPonk Modeling Platform	X	X		X					X	X

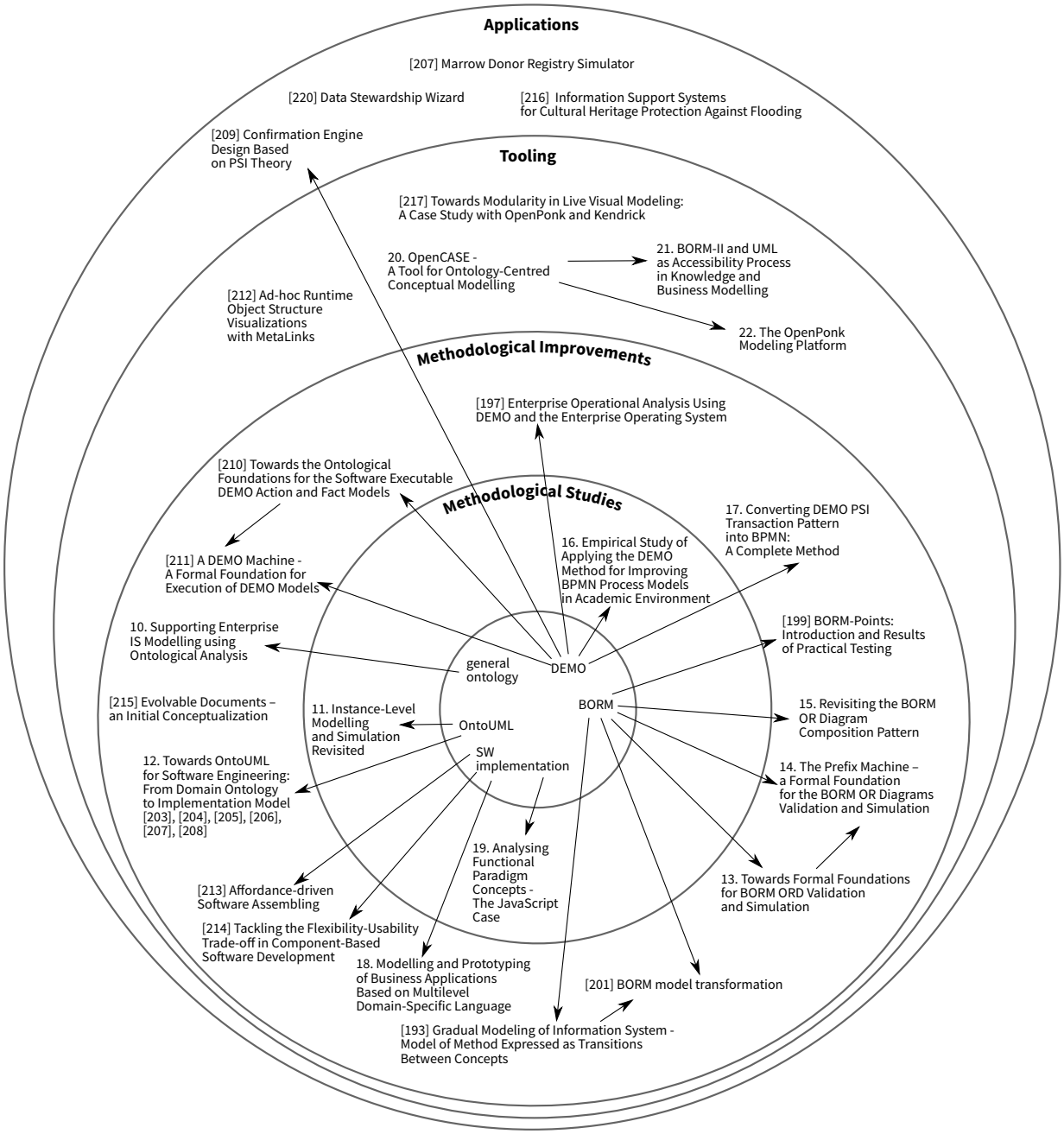


Figure 2.1: Levels of work done

Part I

Introduction

Chapter 3

Conceptualisation

3.1 Introduction

Conceptual modelling (CM) is about "representing aspects of the physical and social world for the purpose of understanding and communication [and, hence] the contribution of a conceptual modeling notation rests in its ability to promote understanding about the depicted reality among human users" [3]. Conceptual modelling is the central topic of CCMi, as it appears to be a missing (highly neglected at best) piece of software and enterprise engineering. The position of CM is depicted in Figure 3.1, which has become CCMi's core scheme.

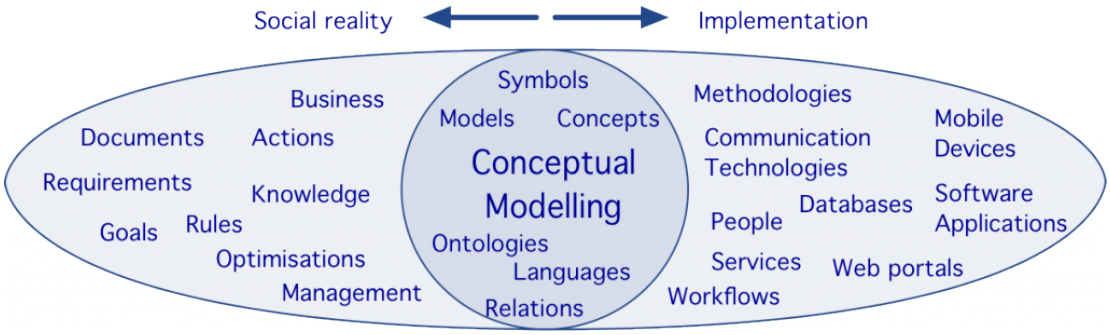


Figure 3.1: The position of conceptual modelling between enterprise and software engineering [2]

At the same time, as will be explained in Chapter 5, conceptualisation in its most general form permeates all human endeavour, so it is present in every place of the scheme. However, if our main goal is harmonisation of enterprise and software engineering, then the conceptualisation embodied in the discipline of conceptual modelling sits in between, as depicted.

3.2 Formal Foundations

Formally, conceptual modelling is based on the **Triangle of Reference** (also called Ogden's Triangle or Semantic Triangle) (Figure 3.2), which was used by Ullmann in his interpretation work [4] (Figure 3.3). The "represents" relation concerns the definition of language real-world semantics. The dotted line between language and reality in this figure highlights the fact that the relation between language and reality is *always intermediated by a certain conceptualisation* [5]. As such, conceptualisation permeates all human intellectual endeavours.

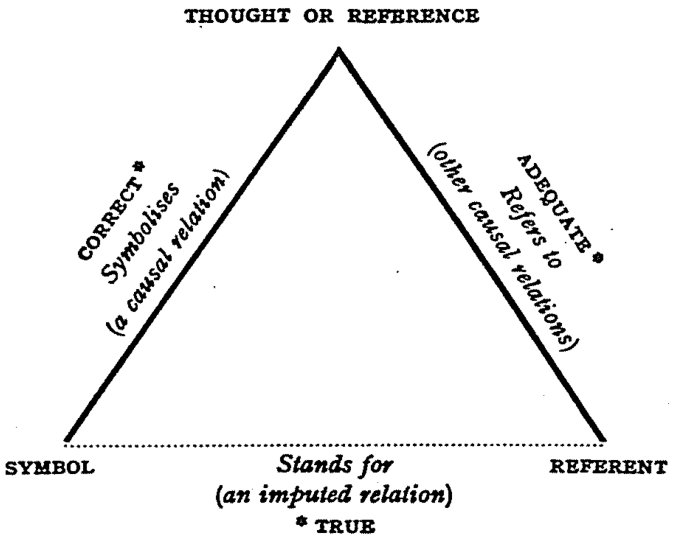


Figure 3.2: The Triangle of Meaning by Ogden [6]

Conceptualisations are abstract entities that only exist in the mind of a user or a community of users of a language. In order to be documented, communicated and analysed, they must be captured, i.e. represented in terms of some concrete artefact. This implies that a language is necessary for representing them in a concise, complete and unambiguous way. [7]

The **semiotic ladder** is an important corner-stone of information science and conceptual modelling. Its original form contains three steps — Syntactics (also called Syntax), Semantics and Pragmatics. Today, its 5-step form enhanced by Stamper is used in enterprise engineering (Figure 3.4).

Mereology is a part of general systems theory on system decomposition and parts, wholes and boundaries (e.g. [9]).

Next, various mathematical disciplines and apparatus are used in conceptual modelling and underlying studies [10], [188], typical representatives being:

- Set theory

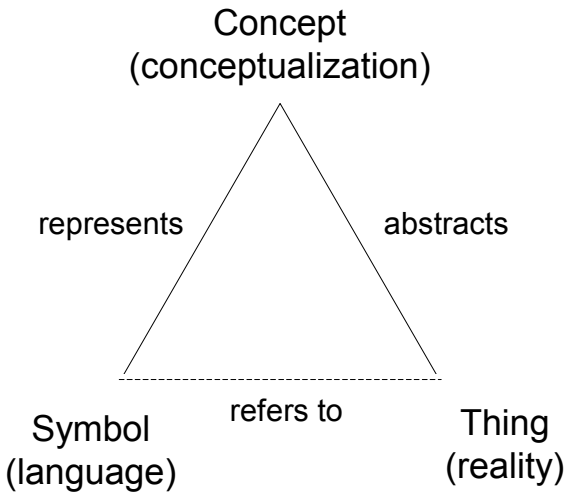


Figure 3.3: Ullmann's Triangle [7]

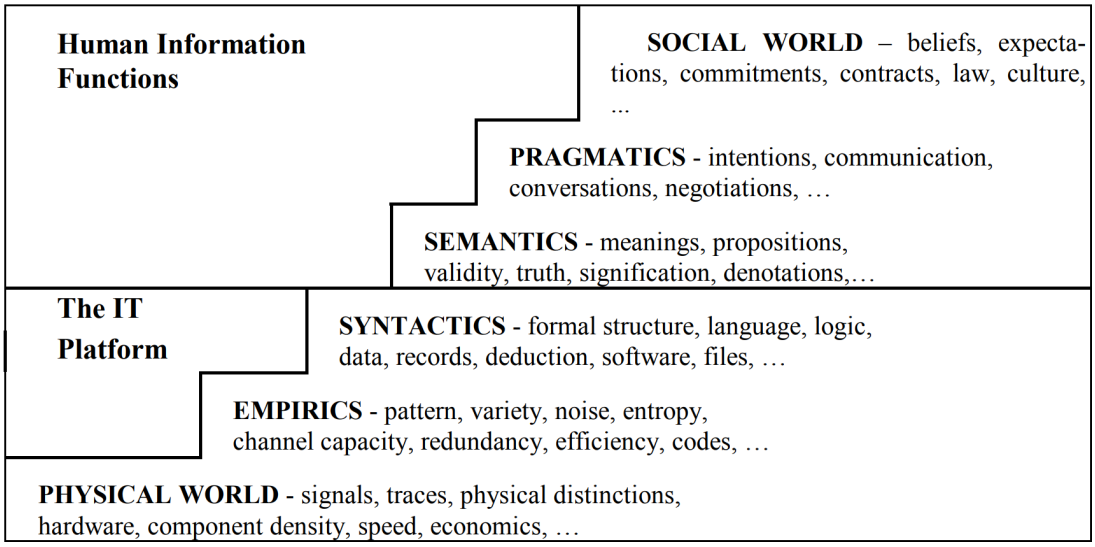


Figure 3.4: Stamper's Semiotic Ladder [8]

- Relations
- Graph theory
- Category theory
- Algebras
- Formal languages and grammars
- Logic: mostly predicate logic, modal logic and various descriptive logics

However, a conceptual modeller must be cautious, as mathematical constructs can lead to *ontologically extravagant* models [7], an example being an empty set containing an empty set.

As for the conceptualisation of behaviour, *finite state machines* and *Petri Nets* are typical formalisations used in information and computer science [188].

3.3 Modelling

Model is an abstraction of a certain state of affairs expressed in terms of a set of domain concepts, i.e., according to a certain **conceptualisation** [7]. As a concrete artefact, a represented model must be expressed in some "suitable"[1] language. The relation between conceptualisation, model, modelling language and specification is depited in Figure 3.5.

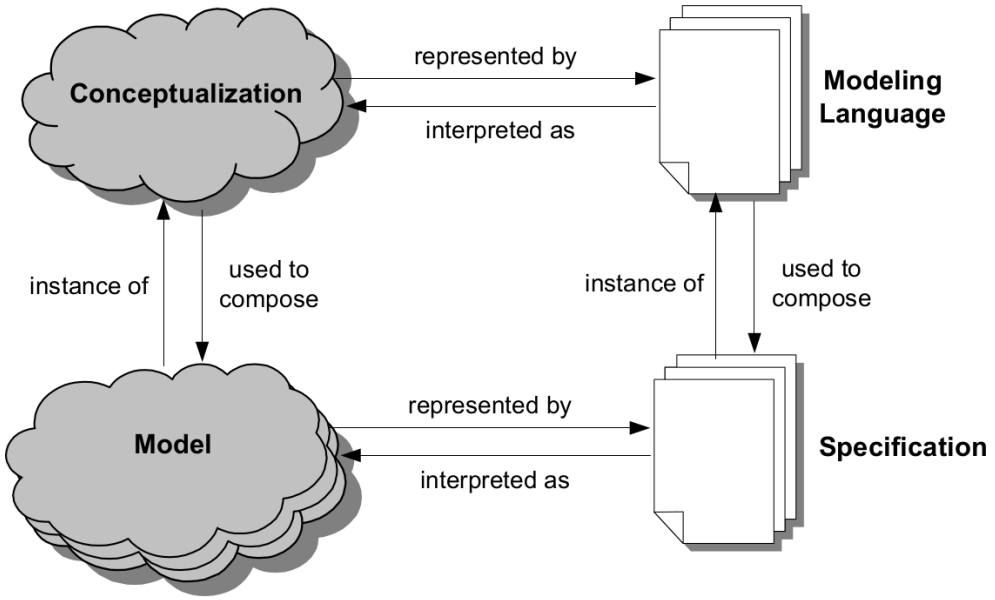


Figure 3.5: Relations between conceptualisation, model, modelling Language and specification [7]

The modelling language may be a **domain-specific language** [11, 12, 13], whose concepts are "tailored" for a certain domain. An example of such a language is DEMO, which is focused on making ontological models of enterprises (Chapter 4), or the work of our student Marek Hakala [259], who created a visual language for modelling of IT architectures.

General conceptual languages are domain-independent languages specifying primitives usable in various domains. An example of such a language is OntoUML discussed in Section 3.4.1.

[1]The suitability here refers to the relevance and language qualities. We do not dive into this topic here, a thorough discussion may be found in [7].

Ontology-driven conceptual modelling (ODCM) [14] is a branch of conceptual modelling that devotes itself to an ontological commitment (Section 3.4). While using mathematical abstractions can lead to conceptualisations that are not aligned with cognitive science [7], ODCM employs sound ontological foundations that are reported to lead to higher-quality conceptual models [189].

3.4 Ontologies

Ontology comes from Greek *ontos* (existence, reality) and *logos* (reason, lore). Philosophical ontology is the science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality. [15]

The term "ontology" (or ontologia) was itself coined in 1613, independently, by two philosophers, Rudolf Gockel (Goclenius), in his *Lexiconphilosophicum* and Jacob Lorhard (Lorhardus), in his *Theatrum philosophicurn.* Its first occurrence in English as recorded by the OED appears in Bailey's dictionary of 1721, which defines ontology as "an Account of being in the Abstract". It was further popularised by Christian Wolff in his work *Philosophia prima sive Ontologia* [16] (Figure 3.6). Regardless of its name, what we now refer to as philosophical ontology has sought the definitive and exhaustive classification of entities in all spheres of being. It can thus be conceived as a kind of generalized chemistry. The taxonomies which result from philosophical ontology have been intended to be definitive in the sense that they could serve as answers to such questions as: "What classes of entities are needed for a complete description and explanation of all the goings-on in the universe" Or: "What classes of entities are needed to give an account of what makes true all truths?" They have been designed to be exhaustive in the sense that all types of entities should be included, including also the types of relations by which entities are tied together. [15].

In engineering disciplines, ontologies became a foundation for **knowledge engineering** [17, 18, 19], software engineering [20, 21, 22, 23, 24] and enterprise engineering [25, 26].

The "modern" definition of ontology comes from Studer [19]: "An ontology is a formal, explicit specification of a shared conceptualisation." This definition is then elaborated on by Guarino in [27].

Ontologies can be *informal* and *formal* (Figure 3.7). While informal ontologies are common and widely used, they tend to be vague, their soundness cannot be guaranteed, they lack inference capabilities and machine-actionability. This is why we focus on formal ontologies in our endeavours, mostly UFO (Section 3.4.1) being a general ontology and DEMO (Chapter 4) being a domain-specific enterprise ontology[2,3].

Although ontological work may seem academic, the truth cannot be further, as Collier puts "the opposite of ontology is not non-ontology, but bad ontology" [29]. In other words, any representation system that has some real-world semantics (as opposed to mere formal semantics) makes an ontological commitment [30].

[2]General vs domain-specific is analogous to the discussion in Section 3.3.

[3]Both are grounded in modal logic, so they belong to the ultimate right on the axis in Figure 3.7.

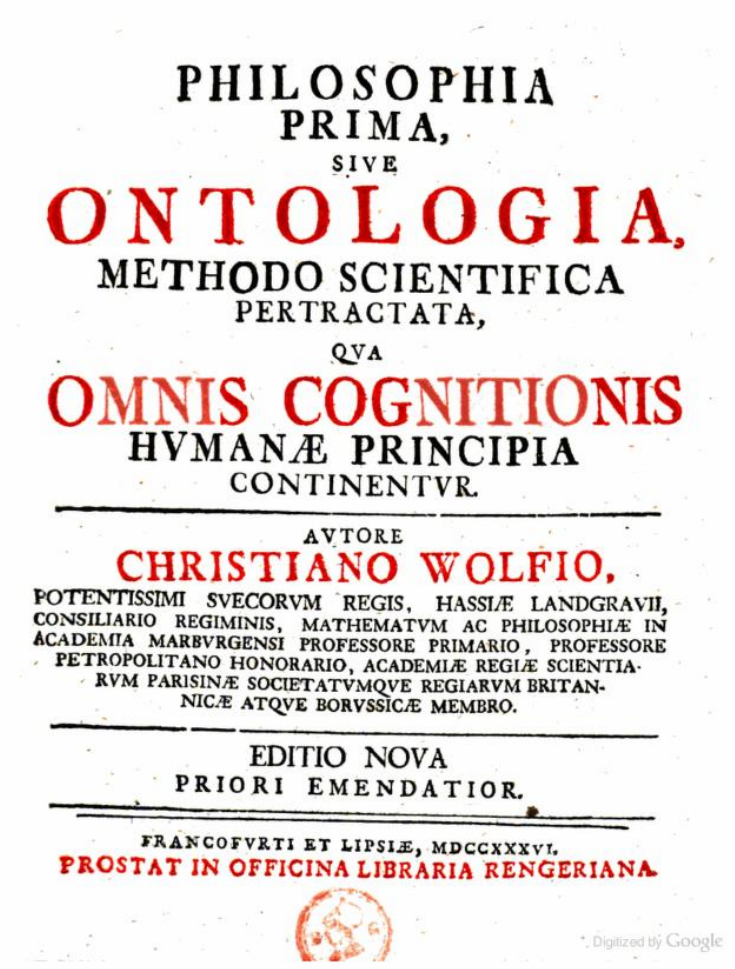

PHILOSOPHIA
PRIMA,
SIVE
ONTOLOGIA,
METHODO SCIENTIFICA
PERTRACTATA,
QVA
OMNIS COGNITIONIS
HVMANÆ PRINCIPIA
CONTINENTVR.
AVTORE
CHRISTIANO WOLFIO,
POTENTISSIMI SVECORVM REGIS, HASSIÆ LANDGRAVII,
CONSILIARIO REGIMINIS, MATHEMATVM AC PHILOSOPHIÆ IN
ACADEMIA MARBVRGENSI PROFESSORE PRIMARIO, PROFESSORE
PETROPOLITANO HONORARIO, ACADEMIÆ REGIÆ SCIENTIA-
RVM PARISINÆ SOCIETATVMQVE REGIARVM BRITAN-
NICÆ ATQVE BORVSSICÆ MEMBRO.
EDITIO NOVA
PRIORI EMENDATIOR.
FRANCOFVRTI ET LIPSIÆ, MDCCXXXVI.
PROSTAT IN OFFICINA LIBRARIA RENGERIANA.
Digitized by Google

Figure 3.6: Philosophia prima sive Ontologia (digitalised by Google)

Although the term "ontology" does not appear in computer and information science frequently, it is due to say that the idea that data are "fragments of a theory of the real-world and data processing is about manipulating models of such a theory" was conveyed by Mealy in 1967 [31]. The paper also directly mentions "ontology". Similarly, Bill Kent in his book "Data and Reality" [32] states that "the questions aren't so much about how we process data as about how we perceive reality" and he puts forward typical ontological challenges such as topics of identity, unity and classification and relationships.

Although we deal with UFO and OntoUML, there are other approaches that have their significance in practice and history. These are notably BWW (Bunge-Wand-Weber) [33], GFO (General Formalized Ontology)/GOL (General Ontology Language) [34] and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [35]. As we find UFO and OntoUML most aligned with our purposes, we do not elaborate on the other alternatives here.

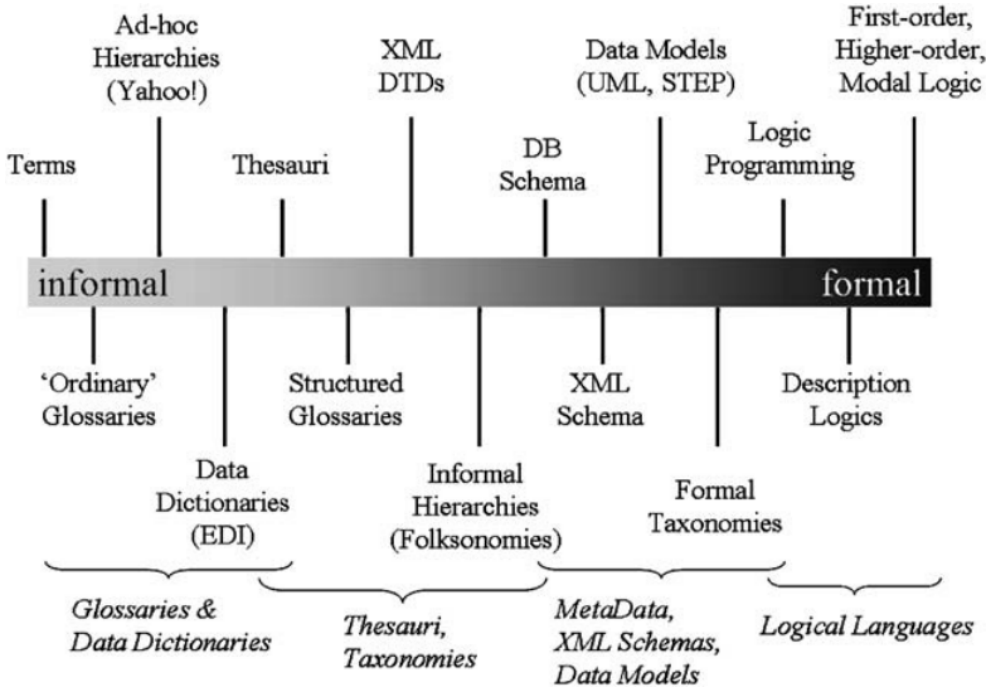


Figure 3.7: Informal and formal ontological approaches [28]

3.4.1 UFO and OntoUML

Unified Foundational Ontology (UFO) originated as an initiative to develop sound ontological foundations for conceptual modelling languages [36, 37].

The UFO ontology was developed by consistently putting together a number of theories originating from areas such as Formal Ontology in philosophy, cognitive science, linguistics and philosophical logics. It comprises a number of micro-theories addressing fundamental conceptual modelling notions. The ontology is divided in three strata dealing with different aspects of reality, namely:

UFO-A — An Ontology of Endurants dealing with aspects of structural conceptual modelling. It is organized as a Four-Category ontology comprising theories of Types and Taxonomic Structures [38, 39] connected to a theory of object identifiers (including a formal semantics in a Sortal Quantified Modal Logics [40]), Part-Whole Relations [41, 42, 43, 44], Particularized Intrinsic Properties, Attributes and Attribute Value Spaces [45, 46, 47] (including a theory of Datatypes as Measure Structures [48]), Particularized Relational Properties and Relations [49, 50, 51] and Roles [52, 53];

UFO-B — An Ontology of Perdurants (Events, Processes) dealing with aspects such as Perdurant Mereology, Temporal Ordering of Perdurants, Object Participation in Perdurants, Causation, Change and the connection between Perdurans and Endurants via Dispositions [54];

UFO-C — An Ontology of Intentional and Social Entities, which is constructed on top of UFO-A and UFO-B, and which addresses notions such as Beliefs, Desires, Intentions, Goals, Actions, Commitments and Claims, Social Roles and Social Particularized Relational Complexes (Social Relators), among others [55, 56].

Domain-specific ontologies are then formulated above these, such as UFO-S (the ontology of services) [57], UFO-L (the ontology of the legal core) [58], the ontology of software engineering [55] or measurement [59].

Then, **OntoUML** [7] was conceived as an ontologically well-founded version of the UML 2.0 fragment of class diagrams, technically being a UML profile (Section 3.6.1), for structural conceptual modelling. The language metamodel strived to incorporate formal syntactical constraints that would delimit the set of grammatically valid models of the language to those that represented intended state of affairs admitted by the underlying ontology. By doing this, the authors have managed to build a modelling language with explicitly defined formal and real-world semantics [60].

UFO and OntoUML have been applied numerous times in a vast variety of domains. We recommend reading [60] for a comprehensive account.

3.4.2 Our Contribution and Applications

In our research, we have focused chiefly on applying UFO and OntoUML in software engineering, which is described in Section 5.1.1. OntoUML is also used in our projects, such as the Marrow Donor Registry Simulator (Section 3.5.1) and also in students' theses, such as [260] and [261] and others mentioned in other places.

3.5 Behaviour Conceptualisation

While conceptual modelling of behaviour is "just" another facet of conceptual modelling and everything applies to it similarly as to the conceptual modelling of structural aspects, it poses some significant challenges and open topics.

Ontological study of behaviour has been parallel and much intertwined with the study of structural aspects of reality. It is rooted in the notion of *perdurants*, being individuals composed of temporal parts — as opposed to *endurants* which are wholly present whenever they are present. Thus, perdurants are an ontological basis for *events* that unroll in time. A notable work in this area has been done by Guizzardi by formulating UFO-B, a concise Unified Foundational Ontology of events [54], Figure 3.8.

Apart from that, structural ontological models in UFO-A also contain behaviour-related information, e.g. *phases* speak about states through which an entity can possibly go through during its lifetime. However, structural models do not contain details about causation nor any timing information.

The discipline of enterprise engineering deals with behaviour intensively. The ψ-theory (Section 4.2) then conceptualises the behaviour in an enterprise as a sequence of *acts* of

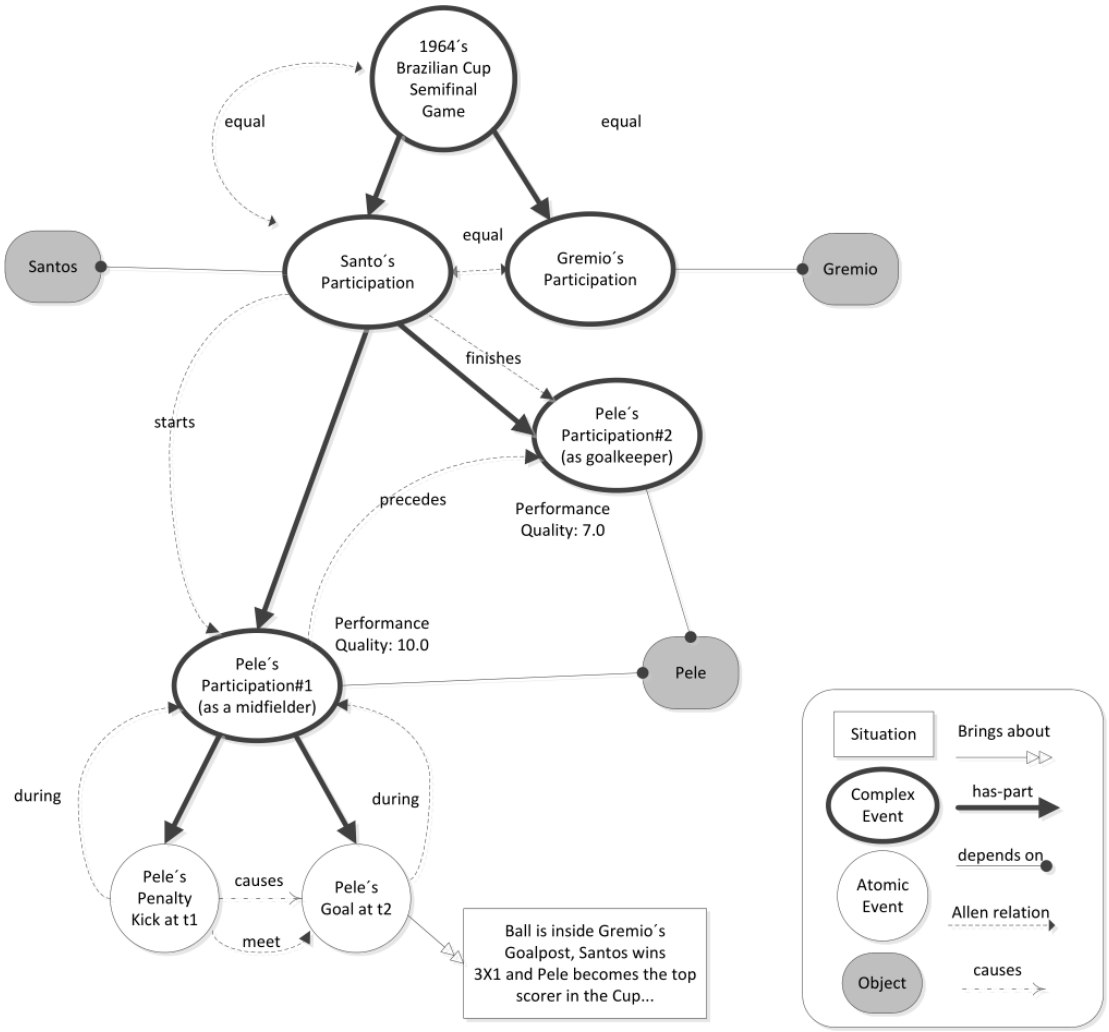


Figure 3.8: A fragment of a football game described with UFO-B [54]

respective actors (Section 4.3.2.2). The view is *event-driven*, i.e. acts occur as a reaction to results of other acts (called facts). The actors then have their *agenda*, being a list of facts that need to be responded to.

Now going from the other side, of technical systems, behaviour conceptualisation has been always one of the most important topics. Each programming paradigm defines an ontology of behaviour (Section 5.2) that is focused on *computation*. Going up with abstractions, we arrive to various notions of *machines* — state machines, Turing Machine, RASP machine and also Petri nets [188]. While these abstractions have been successfully used to describe and reason about computational topics, they are limited in their ability to provide a faithful ontological description of perdurants [190].

3.5.1 Our Contribution and Applications

During our research and practice we identified some open problems in the area of behaviour conceptualisation.

3.5.1.1 Process Modelling Notations

A detailed comparison of BORM (section 4.4), BPMN (section 3.6.2) and DEMO (section 4.3) notations is presented in master's thesis of Vejražková [61], Cimpl then focused on UML (section 3.6.1) and BPMN in his bachelor's thesis [262].

The most used notations for behaviour modelling, UML and BPMN exhibit serious ontological flaws (Section 3.6), while there is still no established process-modelling technique based on UFO-B. The author created a proposal of such a method in the *Marrow Donor Registry Simulator* project [191] (Figure 3.9). The proposed method and visualisation was then used in the bachelor's thesis of Ondřej Král [225].

3.5.1.2 Modularisation

Behaviour modelling notations lack proper modularisation constructs. In BPMN, there is the notion of several types of *subtask*, however it can encapsulate just behaviour of one swim line (i.e. actor).

In DEMO models, the behaviour is modularised by the notion of *transactions* that encapsulate coordination and production acts related to one product. Further modularisation is achieved using various levels of models (Figure 4.6). However, there is also a need of arbitrary modularisation levels. This can be achieved using *scope of interest*, however, a proper meta-model and methodological guidelines are missing[4].

We addressed the topic of modularisation of behaviour with Podloucký in [192] (Chapter 15) for the case of the BORM OR Diagrams, where we proposed two dimensions of modularity: vertical and horizontal, interestingly coming to diagrams strongly resembling UML behaviour models (Figure 3.10).

3.5.1.3 Gradual Transformation of Concepts

In an ideal model-driven engineering world (Section 5.1.1), we would have models of several layers, where the top layer would be a high-level abstraction suitable for managers, the lower levels would then contain details of the domain and would be suited for domain experts and analysts, while the lowest levels would contain more and more technical details, up to an executable artefact. Concepts at various levels would be clearly related, thus supporting verification of completeness, change management and audit trail. We sketched this vision together with Pícka in [193]. While this "dream" has partially happened for structural models, it seems to be far from realisation in behavioural models.

[4] A paper discussing this was presented by Hans Mulder on EEWC 2018, however, the proceedings have not been published, yet, at the time of writing this text.

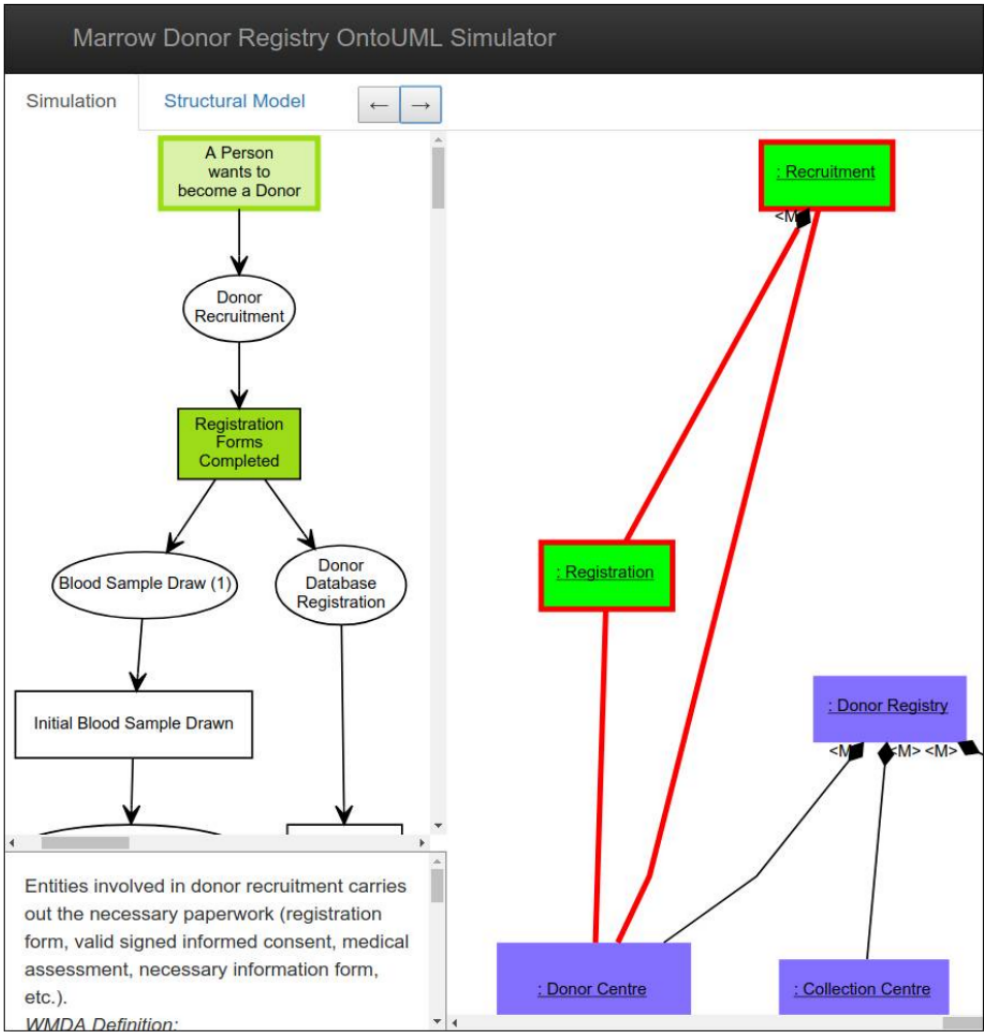


Figure 3.9: Marrow Donor Registry Simulator — process simulation and a dynamically responding instance of the structural model

As the model-driven engineering is introduced in Section 5.1.1 and the topic is highly about implementation, we elaborate on it more in Section 8.2.2.

3.6 Conceptual Modelling Notations

3.6.1 UML and OCL

Unified Modeling Language (UML) [62] is a general-purpose modelling language. As there are common misunderstandings about it's nature and purpose, we take this chance and clarify them.

First, UML is today really a *general purpose* modelling language. Sometimes in litera-

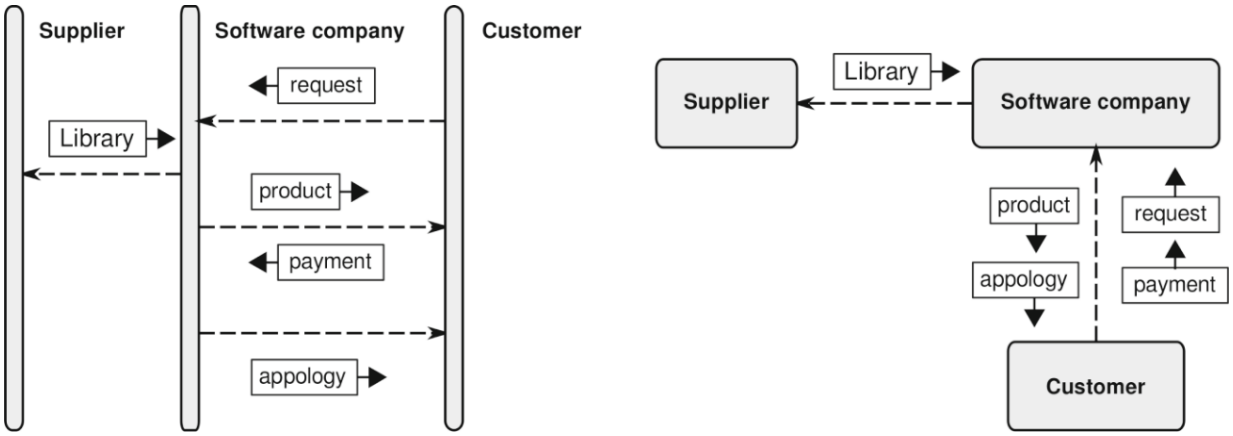


Figure 3.10: Vertical and horizontal decomposition in BORM

ture, it is being introduced as a software-engineering modelling language [63, 64]. While it is true that it evolved from object-oriented modelling methods in 1990s, it is not the essence of UML today [62]. Software engineering diagrams are just one of its *profiles* [65]. Examples of other profiles are the System Modeling Language (SysML, [66]) and OntoUML [67], which are not primarily software-engineering notations.

As such, we can speak about UML either in a *broader sense* — about a meta-modelling language for creating custom domain-specific graphical notations — or in a *narrow sense* about a well-known software-engineering profile, which specifies the most common diagrams used for analysis and design of software artefacts (Figure 3.11).

The second misunderstanding is that UML is a method or even a methodology. It is not, it is just a *notation*, as is clear from its specification [62]. This confusion comes from the fact that the notation is introduced typically in software-engineering books *along* with a methodology ([63, 64]). This lack of a standardised method results also in *different interpretations* of the UML notation and it is hard to use it for sound shared reasoning [68].

Third, although UML in its narrow sense is also used for general-purpose and ontological conceptual modelling, serious flaws in this respect have been identified [69, 68, 70]. It is thus advisable to use ontologically-founded conceptual modelling languages with sound formal foundations, such as OntoUML (Section 3.4.1), whose notation is specified as a UML profile[5].

3.6.1.1 OCL

Object Constraint Language (OCL) [71] is a specification language that is part of the UML standard. It can be used for the following purposes:

- to access model elements and their values,

[5]This is an example of the UML narrow-broad confusion — I even met a renowned scientist in the field of conceptual modelling who, seeing the (negatively) familiar notation, inferred that UFO had the same vague informal foundations.

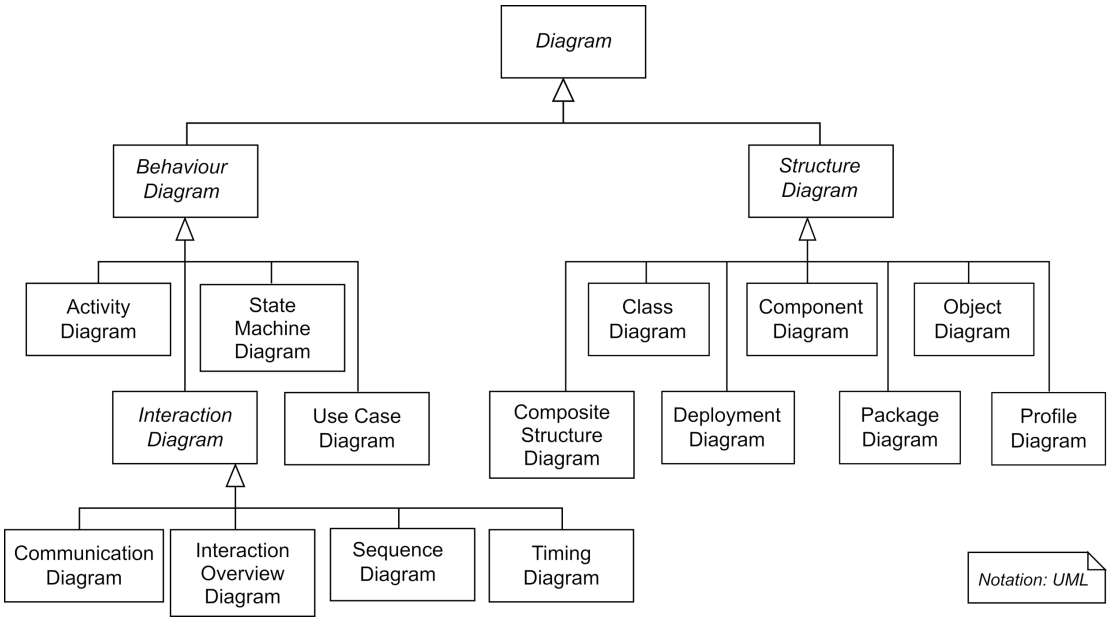


Figure 3.11: UML's software-engineering profile diagrams [62]

- to define constraints and restrictions for model elements and their values,
- and to define query operations [63].

In [72], the authors define basic syntax and semantics of OCL constructs and introduce several tools that support modelling and evaluation of OCL constraints. In [73], the authors define a technique for transformations of OCL constructs into other equivalent forms to support their definition, validation and transformation.

3.6.1.2 Our Contribution and Applications

In our work, we use UML in its broader sense mainly in the form of the OntoUML profile (Section 3.4.1) and its narrow sense to express the object-oriented model in transformations (Section 5.1.1). For this reason, we also intend to have an OCL interpreter in our conceptual modelling platform OpenPonk (Section 8.3.2), however this showed to be a challenging task [226].

We use OCL to enrich structural conceptual models, namely in OntoUML, with domain-specific constraints. This approach is described in [194] (Chapter 11).

3.6.2 BPMN

BPMN stands for "Business Process Modelling Notation" and it is another OMG standard [74]. It originates from the traditional flowcharting notation that was extended with

event-triggered behaviour, exceptions and other constructs; It is also more precisely defined. An example of a BPMN diagram is in Figure 3.12.

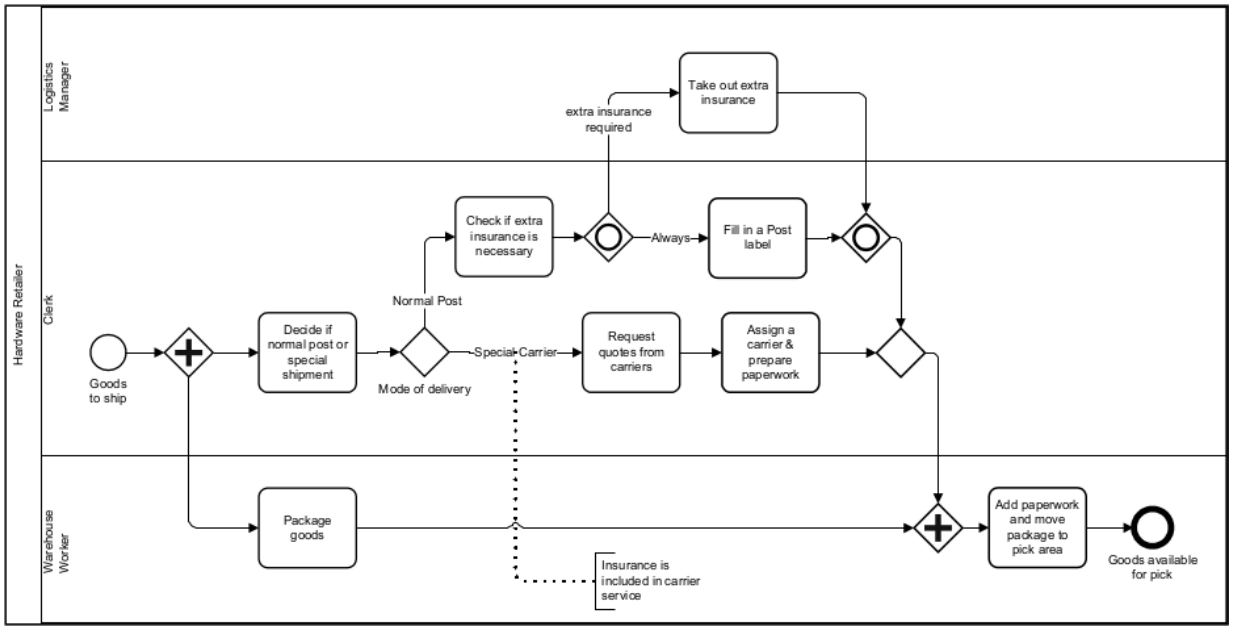


Figure 3.12: BPMN example: Shipment Process of a hardware retailer [75]

A strong point of BPMN is the existence of BPEL, "Business Process Execution Language" by OASIS technical committee, a standard executable language for specifying actions within business processes with web services. BPMN can be translated into BPEL and executed on a workflow engine (Section 8.2.2).

The strong point of BPMN is that being effectively an industrial standard, it has become "linuga franca" in practice and a plethora of commercial and open-source tools is available.

However, BPMN is a mere notation, as the name suggests, so an appropriate modelling method must be engaged, such as Silver's [76]. Silver divides the notation into 3 levels:

Descriptive BPMN suitable for business stakeholders containing just a basic notation elements and a top-down hierarchical style.

Analytical BPMN that uses the full palette of symbols and goes into technical detail of events, exceptions, etc. It represents a common process language shared by business and IT.

Executable BPMN that contains additional technical details to make the process executable on a workflow engine: data, services, messages, etc.

Silver also discusses BPMN flaws that mostly come from the fact that BPMN is a negotiated settlement between rival committees with business interests. Some of them are amended by the method, however there are redundant elements and patterns to express

the same thing, which complicates models interoperability. Some unfortunate diagram patterns are allowed, while some other important constructs are not supported properly (e.g. states).

Further ontological analysis of the notation with respect to simulation models was performed by Guizzardi and Wagner in [77]. Although their preliminary results show 100% soundness, which suggests that the core elements of BPMN have been well-chosen, the results of only 70% lucidity, 60% completeness and 32% laconicity suggest that there are quite a few ambiguous elements, missing concepts, and redundant elements in BPMN.

The solution of this state is twofold:

1. *Replacing BPMN by an alternative* — an example being DEMO (Section 4.3). This approach solves effectively the issues, however, adoption in practice is problematic due to higher methodological complexity and novel notation[6].

2. *Complementing BPMN by an quality-improving method* — an example being the Silver's method or approaches based on applying DEMO methodology for the BPMN notation (Section 4.3.4.1).

[6]Which is a general problem — practice tends to stick to the simplest ways of conceptual modelling possible, in spite of the serious limitations [227].

Chapter 4

Enterprise Engineering

4.1 The Discipline of Enterprise Engineering

A century ago, Taylor published a landmark in the organisational sciences: his Principles of Scientific Management. Many researchers have elaborated on Taylor's principles, or have been influenced otherwise. The authors of the manifesto [78] evaluate a century of enterprise development, and conclude that a paradigm shift is needed for dealing adequately with the challenges that modern enterprises face. Three generic goals were identified:

Intellectual manageability is the basis for mastering complexity. Enterprise phenomena that are not comprehensively understood, cannot be addressed adequately. Hence, the nature of necessary changes cannot be determined; consequently they cannot be brought about effectively. In addition, current development approaches, for enterprises as a whole and for ICT applications in particular, are cursed with combinatorial impacts of changes, which make their implementation slow and practically unmanageable. So, in addition, appropriate ideas of enterprise evolvability are needed for making changes expeditious and manageable — this topic is addressed by the research performed at the University in Antwerp and described in Chapter 6.

Organisational concinnity is conditional for making strategic initiatives operational — in order to perform optimally and to implement changes successfully, enterprises must operate as a *unified and integrated whole*, taking into account all aspects that are deemed relevant. A viable theory and methodology for enterprise engineering must be able to address all relevant aspects, even those that cannot be foreseen presently, in a properly integrated way, so that the operational enterprise is always a coherent and consistent whole. The DEMO methodology (Section 4.3) represents such a way.

Social devotion is the basis for achieving employee empowerment as well as knowledgeable management and governance. Contrary to Taylor's mechanistic view on organisations, enterprise engineering takes a human-centred view. It considers human

beings to be the 'pearls' of every enterprise. Therefore, it suggests that all employees should be fully empowered and competent for the tasks they have to perform. They must be endorsed with transparent authority and have access to all information they need in order to perform their tasks in a responsible way. Next, managers must not only be skilled in managerial work, they must first of all be thoroughly knowledgeable in the subject field of the enterprise they are managing. This is partially addressed in the DEMO method, however, the full span of this topic reaches into managerial, social and psychological sciences.

4.2 Enterprise Engineering Theories

A theoretical framework for positioning the theories, goals, and fundamentals of enterprise engineering is presented in the Manifesto [78], as well. They are categorised into four classes: philosophical, ontological, ideological and technological: Figure 4.1.

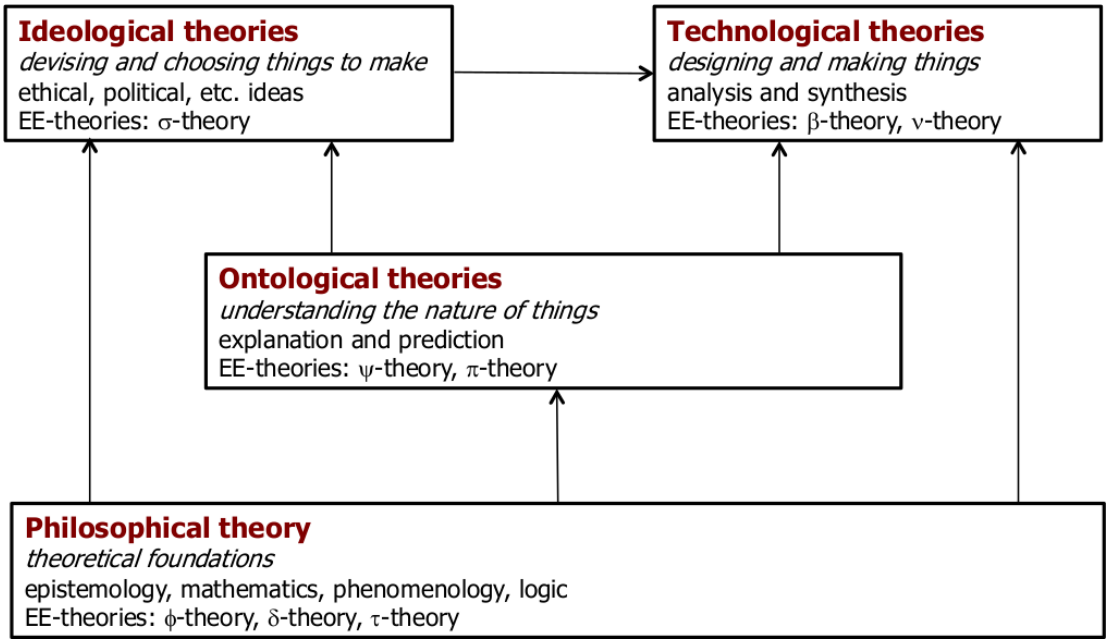


Figure 4.1: Enterprise engineering theories in the classification scheme [78]

An arrow between two classes means that every theory in the class on the arrow side is based on a number of theories (possibly none) in the class on the shaft side.

Philosophical theories are theories that address very basic conceptual matters. They include the philosophical branches of epistemology and phenomenology, as well as logic (in all of its variants) and mathematics. Philosophical theories are valuated by their truthfulness within a chosen area. The truthfulness of a philosophical theory is established by reasoning, and/or by judging its tenability in the face of reality.

Regarding logical and mathematical theories, this reasoning can mostly be exact. In the other branches of philosophy, such exactness is mostly not possible.

Ontological theories are theories about the nature of things. They address explanatory and/or predictive relationships in observed phenomena (Section 3.4). Within the discipline of enterprise engineering, we are particularly concerned with cause-effect relationships in systems. These relationships are (or must be) able to explain observed behaviour, as well as to predict behaviour to some extent, based on the ontological understanding that the theory provides. Ontological theories are valuated by their soundness and their appropriateness. The soundness of an ontological theory is established by its being rooted in sound philosophical theories. The appropriateness of an ontological theory is established by the evaluation of its practical application, e.g., through expert judgments.

Technological theories are theories that address means-end relations between phenomena. Obviously, this is the core area of engineering (of all kinds). Technological theories are the foundation of design methods and methodologies. As Alexander in [1] puts it, a design process is basically a process of analysing a problem (a situation that one considers undesirable) and synthesising a solution (a situation that one considers desirable). After having conceived the solution in all detail, it can be implemented, such that the new situation can be made operational. Implementing is assigning concrete means to the elements of the implementation model. Technological theories are valuated by their rigour and their relevance [79]. The *rigour* of a technological theory is established by its being rooted in sound ontological theories. The *relevance* of a technological theory is established by the evaluation of its practical application, e.g., through measurements, in evaluative comparisons, and in adoption studies.

Ideological theories are theories that address the goals people may want to achieve in society at large, and for us, in enterprises in particular. Ideological theories are fuelled by visions, convictions and beliefs. Therefore, they are by nature subjective, in contrast to the objective ontological and technological theories. The role of ideological theories in enterprise development is to guide the devising and/or choosing of the changes that are considered necessary, and that consequently have to be accomplished. Ideological theories cannot a priori be predicated as truthful or sound and appropriate, nor as or rigorous or relevant, even if they are rooted in rigorous and relevant other theories. One can only speak of their societal significance. The significance of an ideological theory boils ultimately down to its fruitfulness and utility, as determined by its supporters.

The theories have been under intensive development for the last 20 years, being mostly the work of Dietz and Hoogervorst. They were first published in [26] along with the methodological framework and the DEMO methodology. Since then, the newer versions and additional ones have been published as technical reports to foster community feedback [80].

A new edition of the fundamental book [26] is supposed to be published at the end of 2018 containing all the new and updated theories.

4.3 The DEMO Methodology

4.3.1 Methodological Foundations of Enterprise Engineering

Before we focus on the central methodology of enterprise engineering — DEMO — we need to discuss the role of scientific methodologies in enterprise engineering. March and Smith in [81] distinguish between "natural sciences" and "design sciences". *Natural sciences* are concerned with understanding and explaining observable phenomena around us. Examples of natural sciences are physical, biological, social, and behavioural sciences. Specifically regarding enterprises, social and behavioural sciences seek to understand, explain and predict organisational and human phenomena [82]. Therefore, these natural sciences would belong to the class of ontological theories in Figure 4.1. The other important scientific domain is identified as *design sciences* [83] and they are concerned with devising artefacts or other intentionally created results. Therefore, these sciences would belong to the class of technological theories in Figure 4.1. To further illustrate the distinction between natural sciences and design sciences, one might say that natural sciences are about finding out how things are, whereas design sciences are about finding out what is effective [82]. Put differently, design sciences are about prescribing how things have to be created [81].

Many approaches concerning enterprise design can be noticed with a focus on models and representations, whereby adequate attention to the theory base can be questioned [84, 77, 85]. The so-called **design science research (DSR)** methodology seems an appropriate candidate for being the main research methodology in enterprise engineering [78]. It is also already quite widely accepted, notably in the information systems area (see [81, 82, 86]).

4.3.2 DEMO Principles

DEMO, Design & Engineering Methodology for Organisations, was conceived as an engineering methodology firmly based on the EE theories and supporting the EE principles and fundamentals [26, 78]:

4.3.2.1 Strict Distinction Between Function and Construction

This fundamental is rooted in the τ-theory. In this sense, DEMO provides a *construction* view of the enterprise — its models are white-box and objective.

4.3.2.2 Focus on Essential Transactions and Actors

The complexity of enterprises necessitates a division of tasks to be performed. Because the enterprise must operate as a unified whole, task differentiation must be properly paired

to the integration of the distinct tasks. This is the core of the ψ-theory (Performance in Social Interaction). The notion of differentiation implies that employees are engaged in numerous different production activities (e.g., concluding an insurance policy, making an equipment part, paying an invoice, or giving a permission), whereas the notion of integration demands that these activities are coordinated such that the enterprise operates as an integrated whole. The ψ-theory provides us with the insight that the coordination and production activities occur in universal patterns called **transactions** (Figure 4.2). These are the elementary (essential) organisational building blocks of enterprises. Enterprises have dozens of processes, such as for production, recruitment, purchasing, payment, accounting, logistics, and so on. Despite their different nature, they all share the same underlying transaction patterns. Every business process appears to be a tree structure of transactions (Figure 4.3).

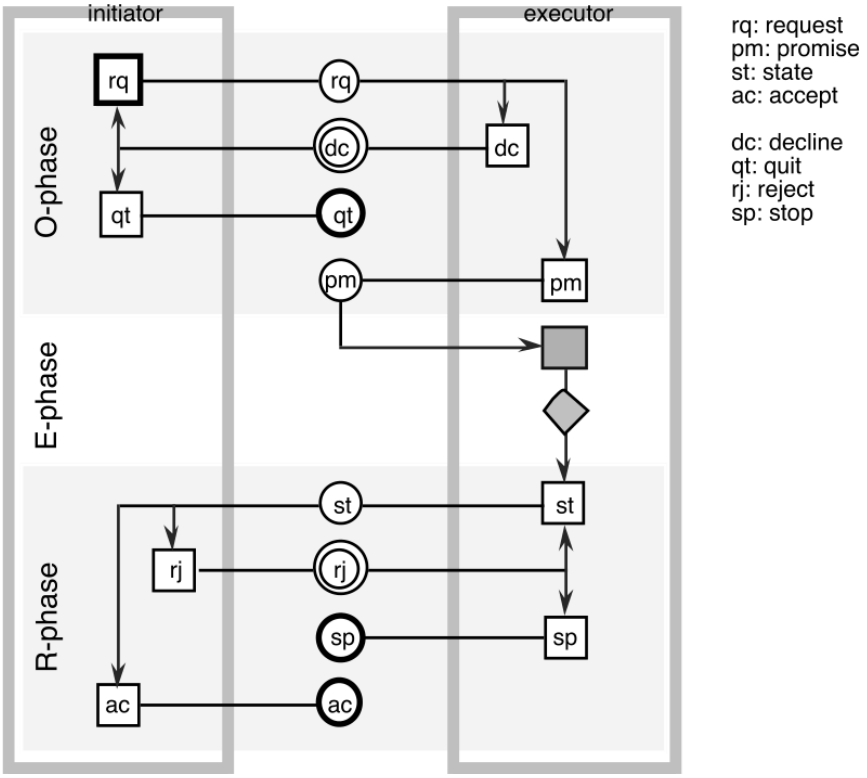


Figure 4.2: The Standard Transaction Pattern [26]

Another major contribution of the ψ-theory to mastering the complexity of organisations emerges from the distinction between an enterprise's B-organisation (from business), I-organisation (from information), and D-organisation (from data and document) [26]. The ψ-theory-based ontological model of the B-organisation of an enterprise is called its **essential model**. By adopting this distinction, next to the organisational building block from the ψ-theory, a reduction in the size of enterprise models is achieved, and in the time to produce them, of well over 90% [87]. Consequently, a major contribution is offered

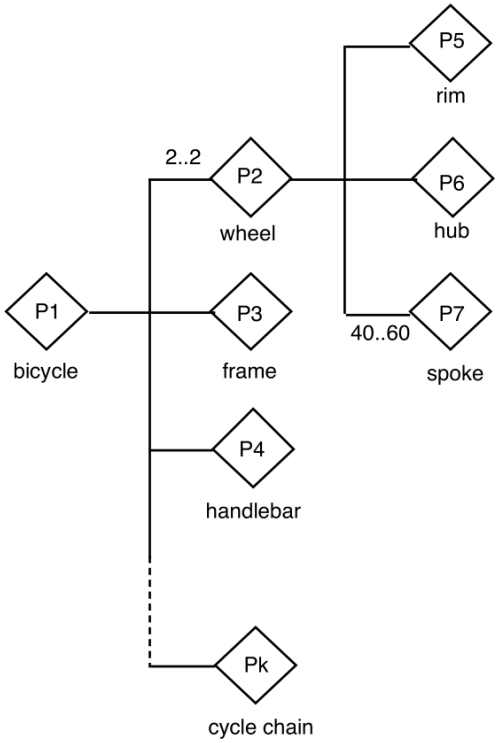


Figure 4.3: Transaction tree of manufacturing a bicycle [26]

to making enterprises intellectually manageable. Focusing just on the B-organisation is called **abstraction from realisation** — the I-organisation being then the realisation of the B-organisation and the D-organisation being the realisation of the I-organisation (Figure 4.4).

4.3.2.3 Rigorous Distinction Between Design and Implementation

The β-theory fully explains and clarifies the complete development process of a system, consisting of three phases: (i) function design, (ii) construction design, and (iii) engineering (also called implementation design) [88]. By **implementation** is meant the concrete realisation of a system. Put differently, implementation concerns the activities for putting a design into effect. Unlike the other two phases, engineering is a rather deterministic process executed according to some plan: a precisely defined, detailed scheme of activities, for accomplishing a clearly defined objective (MDE, Section 5.1.1, being a concrete example from software engineering). Contrary to engineering, design is a highly non-deterministic process. It amounts to unrestrictedly exploring design possibilities rather than restrictedly following a predefined, formalised plan. In the function design phase, the functional (black-box) model of the object system, i.e., the system to be developed, is produced, starting from the given functional requirements and the functional principles in the applicable architecture. Ideally, the functional requirements are based on the essential model of the

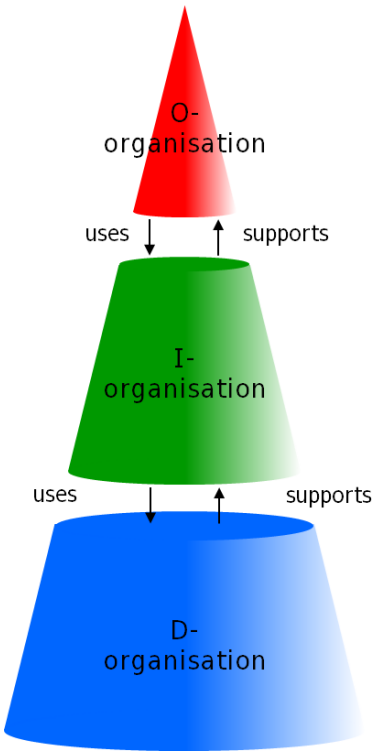


Figure 4.4: The Realisation of an Organisation [26]

using system, i.e., the system that is going to be supported by the object system. In the construction design phase, a highly abstracted constructional model of the object system is produced, starting from the functional model. Ideally, this abstract model is an ontological model, which means that it is fully independent of the way it is or will be implemented. Consequently, design activities cannot be executed and managed as a project. Applying implementation-type concepts to design activities amounts to confusing creativity with execution and planning [78]. In terms of the notion of system lifecycle, enterprise engineering is concerned with all activities up to the implementation stage, as defined above. Utilisation of the (implemented) system pertains to the operational utilisation of the system, which also includes support activities such as maintenance. The overall schema as presented by the β-theory is presented in Figure 4.5.

In the end, essential models need to be realised and implemented, for which much more detailed models have to be produced, guided by enterprise architecture. For software engineering implementation, MDE methods have been developed [89, 90] and also direct execution of the DEMO models [91, 92].

4.3.2.4 Other Fundamentals

Other fundamentals described in the manifesto [78] are related more to managerial and social aspects of enterprise engineering, so we do not present them here.

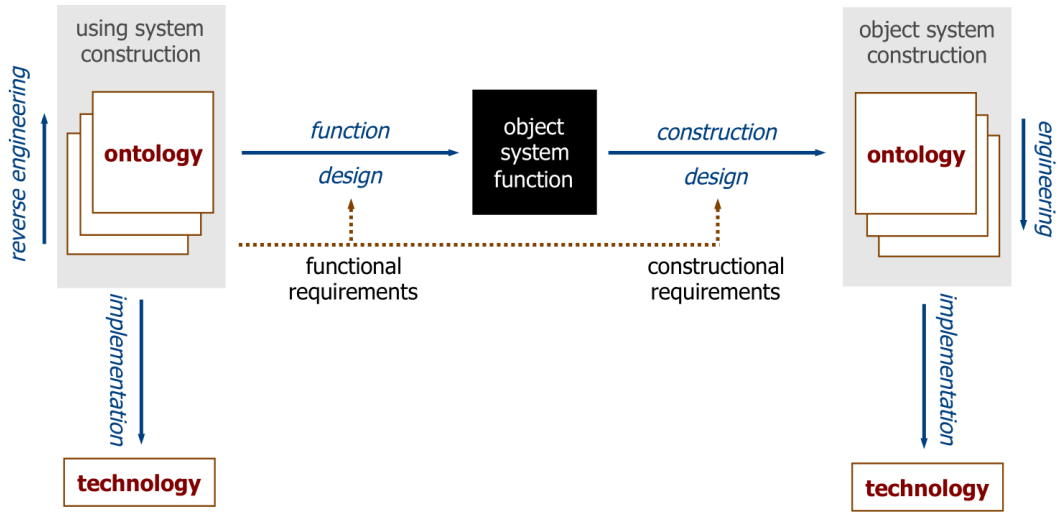


Figure 4.5: Generic System Development Process as specified by the β-theory [88]

4.3.3 DEMO Diagrams

DEMO consists of 4 aspect models, which are tightly related by a common meta-model (Figure 4.6):

Construction model (CM) — the most high-level model depicting transaction kinds, their products and actor roles.

Process Model (PM) — this model brings details about the process consisting of coordination acts causal relations.

Fact model (FM) — a structural model of products related to transactions and other facts.

Action Model (AM) — a textual model containing detailed coordination and production rules.

A detailed description of the diagrams is not the goal here, we refer the reader to the bibliography and also teaching materials of the MI-MEP course. Some basics are also explained in the respective chapters. Let us just comment that compared to e.g. UML models (Section 3.6.1), we can observe tighter inter-relations between the models and a careful complexity management (depicted by the size of compartments in Figure 4.6). Most important, all the diagrams are mere views depicting a certain facet (fragment) of the whole fact-based model. This enables to build tools with advanced verification and auxiliary functions, which is an initiative that starts to take place in the community (see Section 4.3.4).

Another remark should be given about the notion of process in DEMO. Although there is a process-flow (workflow) happing in the end in an implementation, the models are not

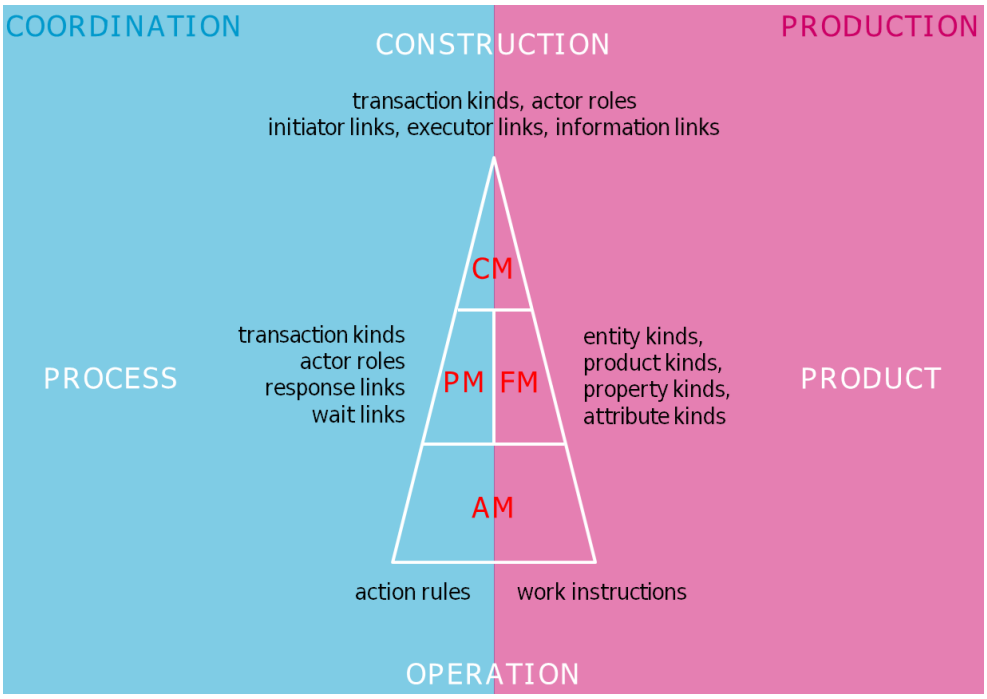


Figure 4.6: Four aspect DEMO models (according to [26])

flow-based (as compared to e.g. BPMN, Section 3.6.2), but rather event-based, the events being coordination facts that trigger successive acts. This conceptualisation enables to express much more complex business rules, however it is challenging for comprehension, especially by untrained stakeholders [263, 264].

4.3.4 Our Contribution and Applications

CIAO! Enterprise Engineering Network [93] was founded to gather the enterprise engineering community. It currently consists of 13 institutes world-wide that share the vision of enterprise engineering according to the manifesto [78] and committed themselves to teaching, practice and research of the theories and methodologies. The focus spans from management (e.g. University of St. Gallen), through conceptual modelling (e.g. Luxembourg Institute of Science and Technology) to implementation (e.g. University of Antwerp or University of Madeira), thus covering most of the domains depicted in Figure 3.1. As such, CIAO!'s philosophy and approach is highly aligned with CCMi's.

We have been active in several research topics:

- Applying the DEMO methodology for improving business process management (Section 4.3.4.1)
- Using DEMO for Smart Contracts specification (Section 4.3.4.2)

- Execution of DEMO models (Section 5.1.2.1)
- Evolvability (Section 6.5)
- Applying enterprise engineering theories for component software systems design (Section 6.5.1)

Apart from that, we have been using DEMO in our projects, such as the CTU data warehouse [228], [265] and it has been also applied in students' theses such as [229].

4.3.4.1 Applying the DEMO methodology for improving business process management

The DEMO methodology not only offers modelling notations, but it also provides an elaborated methodology for complexity management and ontological qualities (Section 4.3). Along with others, we have shown that they can be used for improving current business process management techniques and notations, in our case we performed a study in an academic environment ([195], Chapter 16).

As BPMN is considered an industrial standard for business process modelling (Section 3.6.2), we explored the possibilities of converting DEMO diagrams into BPMN. The foundations were laid in successful master's theses [264, 266]. In the end, we formulated a unique method for lossless transformation of the complete DEMO transaction pattern into BPMN diagrams ([196], Chapter 17). Although the resulting BPMN models are very complex, as was discussed in [263], their completeness opens a way to bring the precision of DEMO diagrams into BPMN. An automated transformation along with execution in a BPMN engine are promising next steps with this respect.

The above research represents a "top-down" approach in business process management: getting from conceptual models into execution (see also Section 5.1.1). Recently, also the "bottom-up" approach is getting an attention. The rationale is that in enterprise practice, the processes in operation usually diverge from their models, or the models had not been elaborated at all [94]. Starting from analysing the existing processes in operation and then applying enterprise engineering methods seems promising. We pursued this consideration and shown in [197] that DEMO can be used along process mining techniques. This path is now further pursued by Marek Skotnica in his PhD research.

Next, we have been also dealing with accessibility of DEMO for non-technical stakeholders. Adam Žďára designed and implemented interactive visualisations of DEMO models for managers in his master's thesis [230] and Petr Nymsa proposed managerial reports above DEMO in his bachelor's thesis [267].

4.3.4.2 Using DEMO for Smart Contracts specification

This side (but promising) research explores a potential of combining Block Chain technologies together with DEMO. The core is the master's thesis of Barbora Hornáčková [198] supervised by Marek Skotnica. They showed that DEMO transactions may serve as a

promising higher-level description language for Smart Contracts. A paper summarising the results was presented at the Enterprise Engineering Working Conference (EEWC) in June 2018 and it raised an attention in the community.

4.4 Business Object Relation Modelling

Business Object Relation Modelling (BORM) is a complex method for analysis and design of object-oriented systems [95]. It consists of several consequential stages that gradually evolve the design of the system towards its implementation. The development of this method started at the Loughborough University in 1993 as a unified approach to business and IT systems modelling based upon the pure object-oriented paradigm inspired by programming languages such as Smalltalk. Its further development has been carried out with the support of Deloitte Central Europe and it became a practically-applied method ever since [95] in numerous large projects in Deloitte Consulting [96], urban planning [97, 98] and others such as: [99]

- the identification of business processes in Prague city hospitals,
- the modelling of properties necessary for the general agricultural commodities wholesale sector in the Central European region,
- business process re-engineering in the electricity supply industry,
- business process re-engineering for a telecommunication network management in the Central European region.

4.4.1 The Object-Oriented Paradigm Alignment

Business Object Relation Modelling is based on the pure object-oriented paradigm. This paradigm is an antropomorphic approach to conceptualisation of structure and behaviour of systems, mostly exercised in the software programming and analysis (Section 7.2). The paradigm's central concepts are *objects* and *message sending* (Section 7.2). Similarly, BORM focuses on communication between the stakeholders (called participants) and hiding of the state-process inside them.

4.4.2 BORM Object Behaviour Analysis

The initial stage of BORM is called *Object Behaviour Analysis* (OBA). It focuses on modelling the objects' behaviour in the designed IT system, a.k.a. the process analysis [99].

The BORM OBA method is based on the pure object-oriented paradigm described above. The central role is given to **participants**, being the objects as opposed to "process-flow" notion of flowchart-based techniques such as EPCs, BPMN, UML activity diagrams etc. These methods are based on creating one continuous process flow; Boundaries in the

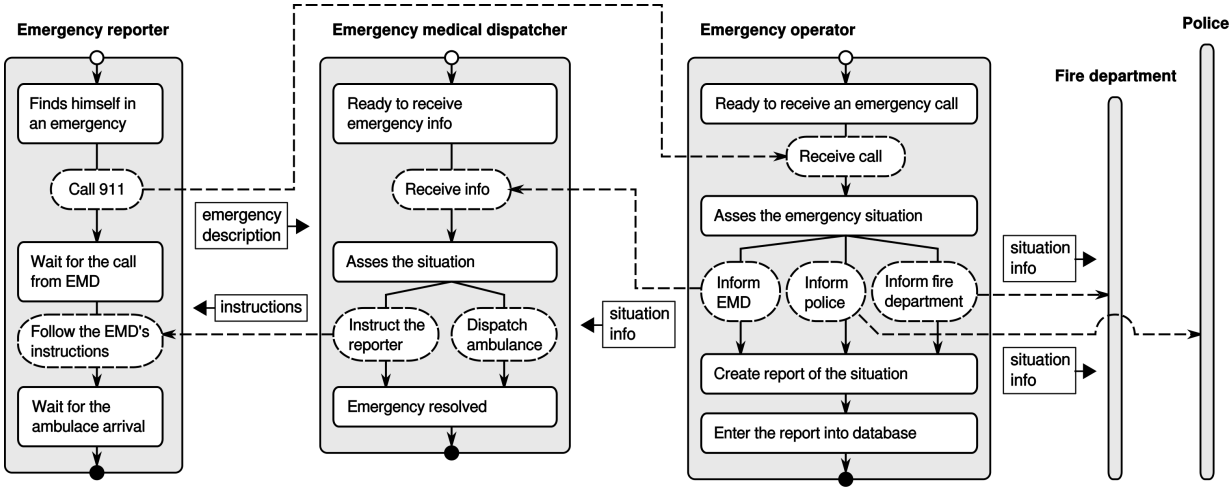


Figure 4.7: A sample BORM Object Relation Diagram.

form of swim lanes can be crossed by the flow, which clearly violates the OO paradigm. In BORM, a participant represents an autonomous object with its own encapsulated inner process flow. Participants are connected just by communications, i.e. sending messages through input and output **data flows**.

4.4.3 Process models in OBA

The central diagram of OBA is the **Object Relation Diagram (ORD)** that captures a behaviour and a communication among objects in the organisation. Figure 4.7 presents a sample ORD. Details of the notation and our modifications to the original one are described in the respective chapters mentioned below.

4.4.4 Our Contribution and Applications

The author was working in Deloitte consulting 2004-2008 and during these years applied BORM on projects for Czech Post and Czech Airlines [199] and was also a member of the Craft.CASE [100] consulting team in its beginnings. Unfortunately, due to the license politic, Craft.CASE stopped to be a choice for conceptual modelling platform at some point, which resulted in own efforts described in Chapter 8.

The core research on BORM in CCMi was the research performed together with Martin Podloucký during his (unfinished) PhD studies. Our goal was to bring sound formal foundations for BORM ORD, which were missing, along with settling execution semantics. As BORM originated as a business-oriented method, one of its main goals was to be simple, which brought a certain degree of vagueness. In our research, we strived to maintain this simplicity, while amending some of the behavioural shortcomings.

The first work [190] (Chapter 13) was focused on identifying the issues in BORM ORD semantics and execution, mostly related to decision making and parallelism. We formulated

the *Simultaneity Principle* and the *Dependency principle* and introduced *Input and Output Conditions* to effectively address the problems.

Next, in [200] (Chapter 14), we formulated the *Prefix Machine*, a formalism bringing precise behaviour specification of ORD. Apart from bringing a formal specification, the Prefix Machine has also other interesting possibilities for process analysis. Using BORM in simulation models was then explored by Veronika Larionová in her bachelor thesis [231].

Realising that *modularity and composition* are key ingredients in managing complexity[1], we then revisited the topic of composition in ORD in [192] (Chapter 15) and formulated the concepts of *horizontal* and *vertical* decomposition (Figure 3.10).

We then returned to the topic of execution semantics and using the Prefix Machine, we formulated a transformation from Coloured Petri Nets into ORD to ground the execution semantics in an existing formal apparatus. Unfortunately, the work has not been finished due to leave of Podloucký and prevailing efforts in other areas (such as Chapter 4 and Chapter 8). However, the research may be revived now in the cooperation with the University of Antwerp, as BORM ORD seems like a possible suitable notation for modelling Normalized Systems behaviour (Chapter 6).

Last but not least, we also put a considerable effort in BORM tooling support, which is described in Chapter 8.

4.5 MEMO

Ulrich Frank and his group at The University of Duisburg-Essen developed a multi-perspective enterprise modelling conceptual framework together with a programming languages called MEMO [101]. It differentiates three so called perspectives — strategy, organization and information system — each of which is structured by four aspects: structure, process, resources, goals (Figure 4.8).

MEMO offers three specialized languages that support the construction of models: The strategy modelling language (MEMO-SML) for strategic planning. The organization modelling language (MEMO-OrgML) serves to model a firm's organization, including business processes and resources. To allow for the specification of information as a foundation of database design or software development in general, MEMO also includes an object-oriented modelling language (MEMO-OML) [101].

MEMO's way of working is through specification of multiple layers of domain-specific modelling languages (DSML) (Figure 4.9) and it exhibits several characteristics:

Flexible Number of Classification Levels — MEMO can model an arbitrary level of classification levels, as is shown in Figure 4.9.

Relaxing the Rigid Instantiation/Specialization Dichotomy — Instantiation can naturally happen at any level of specialisation (dark objects in Figure 4.9).

[1] More in Chapter 6.

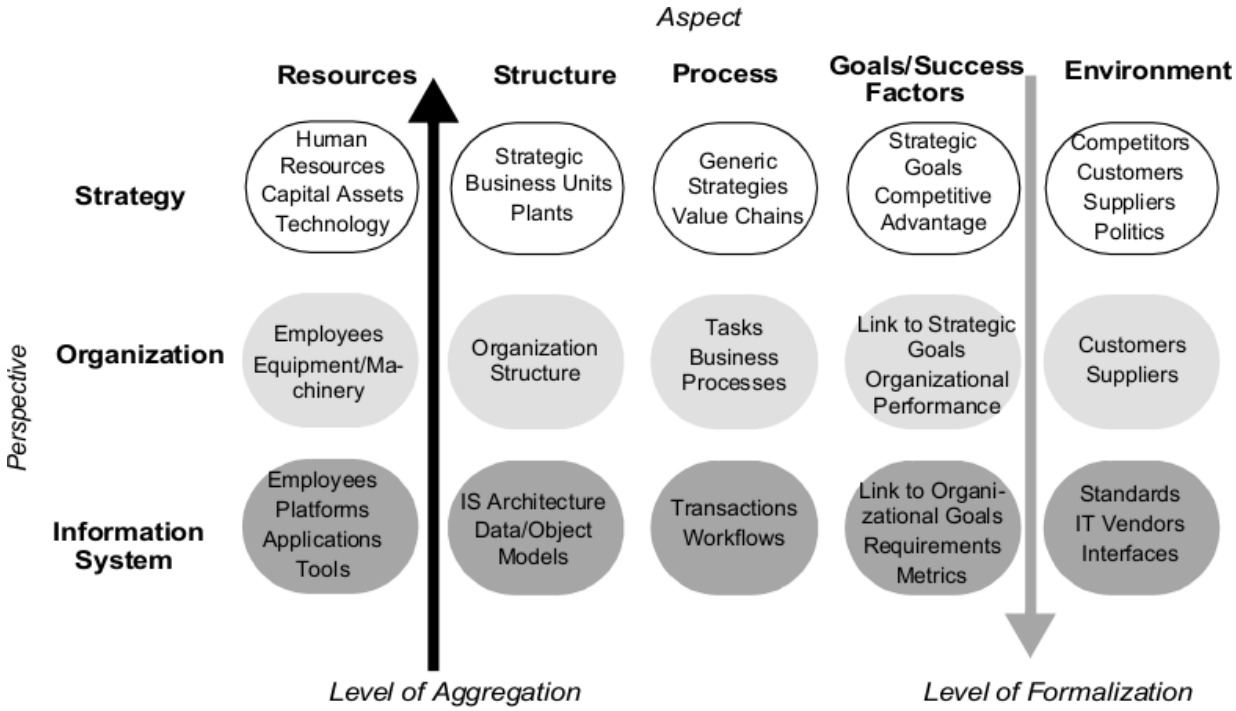


Figure 4.8: MEMO's multi-perspective structre [101]

No Strict Separation of Language Levels — MEMO allows a versatile conception of (meta) models that allow classes on different classification levels to be part of one model. This enables to utter statements such as "Cross Racer R3 is one of our most successful products", while "Cross Racer R3" represents a concept on a different level than "Product".

By committing to such a conceptualisation principles, MEMO effectively solves complexities related to multi-level conceptual modelling that has been traditionally addressed by **powertypes** [103, 104]. However, MEMO approach does not take into account subtler ontological distinctions and aspects, as can be seen from extensive work of Carvalho et al. [105, 106, 107, 108, 109].

As for the implementation of MEMO, the first version of MEMO Tools for modelling, simulation and execution was implemented in Smalltalk (VisualWorks). In 2003 it was reimplemented in Java Eclipse, which enabled better meta-modelling facilities through use of XMF (executable Metamodeling Facility [110, 111]), which enables execution of models (Section 5.1.2). A subset of MEMO was also implemented in the ADO*xx* platform [112].

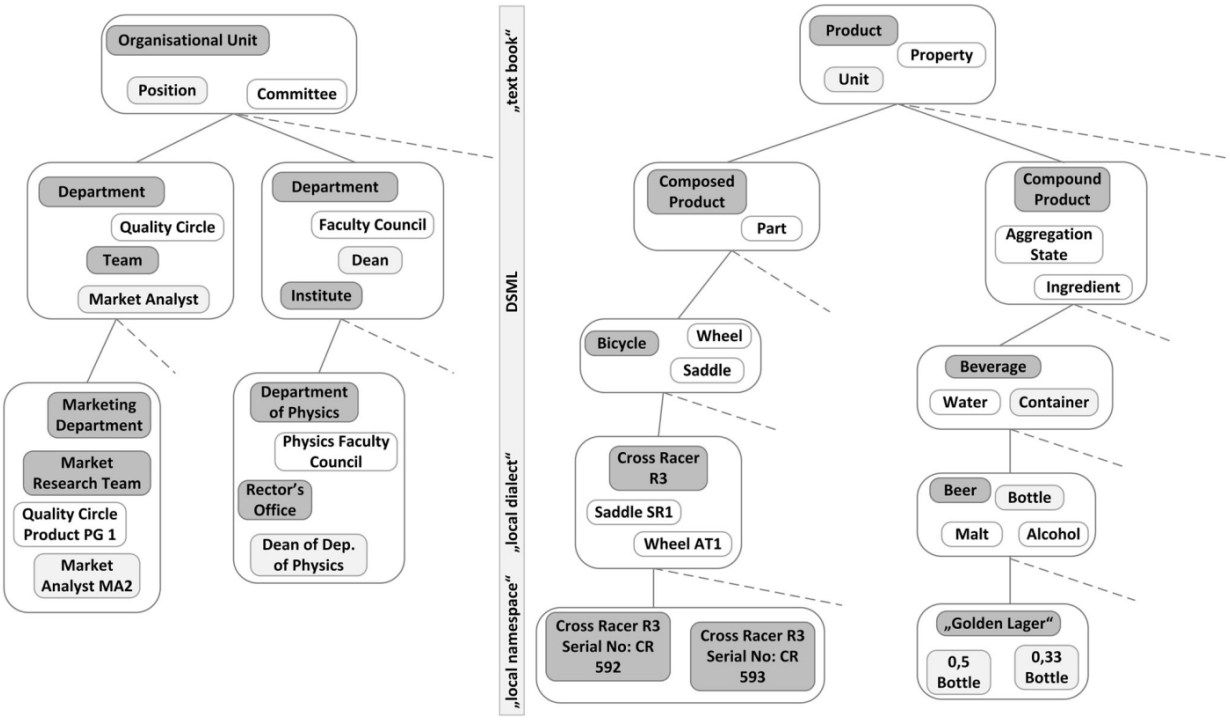


Figure 4.9: An example of a MEMO hierarchy of languages [102]

Chapter 5

The Relation of Conceptualisation and Implementation

There are two types of the relation of conceptualisation and implementation:

1. The relation of conceptual ontological models and their implementation (Section 5.1).

2. Conceptualisation in the implementation itself (Section 5.2).

5.1 The Relation of Conceptual Ontological Models and Their Implementation

Realising the essence of the relation lies in realising the distinction between *ontology* (or "natural sciences") and *design science*, which was explained in Section 4.1 — While ontology deals with "what is", i.e. things that exist (Section 3.4), design science is a discipline of creating artefacts, i.e. things to be. This also comes to a frequent question of the relation between conceptual modelling and ontologies. The answer is that conceptual modelling is a broader term: we may model things that are, i.e. we do a work of an ontologist, or we model the artefacts to be created, i.e. a designer's work. In enterprise engineering, both are usually performed: first we need to describe the current state — usually called the "as-is model" — and then we proceed to reengineering, supportive technical systems design, etc., where "to-be" models are created.

In harmonising enterprise and software engineering, the study of the relation and the supporting methods is an important topic. We can observe two major approaches taken:

1. Model-Driven Engineering

2. Model Execution

5.1.1 Model-Driven Engineering

Model-driven engineering denotes efforts to elevate the level of abstraction used to develop software [113]. One typical effort was the development of computer-aided software engineering (CASE) in 1980s, which focused on developing software methods and tools that enabled developers to express their designs in terms of conceptual models, typically diagrams [114]. CASE tools however did not generally succeed, mostly due to a big semantic gap between the conceptual and technical levels [113].

Today's model-driven engineering efforts are mostly framed by the Model Driven Architecture (MDA)[1], being a framework for software development defined by the Object Management Group (OMG). The key to MDA is the importance of models in the software development process. Within MDA, the software development process is driven by the activity of modelling the software system [115], however, compared to the CASE approach, this happens at several interconnected levels [116], which overcomes the semantic gap between the conceptual and technical levels [117]:

Computation Independent Model (CIM) — A model describing the environment, business processes and business requirements for the product. This model is the most abstract specification dealing with the real business, abstracting from any concrete and specific implementation and technologies. The goal of the model is to capture what is expected of the product. CIM is usually referred to as the *domain model*.

Platform Independent Model (PIM) — A model describing the requirements and specification of the system. The model usually consists of conceptual data models, use case models, description of system functions and processes. However, all the requirements are defined in a general form abstracting from a concrete technologies and platform. The goal of the model is to define the system functions and behaviour that can be applied to various technologies, platforms and environments. PIM is usually referred to as the conceptual[2] or *analytical model*.

Platform Specific Model (PSM) — A model describing the design of the system for a specific platform. This model captures the way how the system requirements defined in the PIM are realized using the specific technologies. Therefore, platform specific tools, constructs, libraries and objects can be used in the model. PSM is usually referred to as the *logical model* of the system and provides visualisation and documentation of the final source code.

Implementation Specific Model (ISM) — This level of abstraction is the actual code of the system with its implementation documentation (JavaDoc, PHPDoc, etc.). Sometimes ISM is also referred to as the *physical model*.

[1] Syntactically, a hyphen between Model-Driven is common. However, in OMG materials, the hyphen is not present.

[2] The attentive reader has probably realised that this name is not fully ontologically correct, as all the four models are in their essence conceptual models according to the definition (Chapter 3).

The relation between the models is shown in Figure 5.1. The software development method based on MDA is called the **Model-Driven Development (MDD)**.

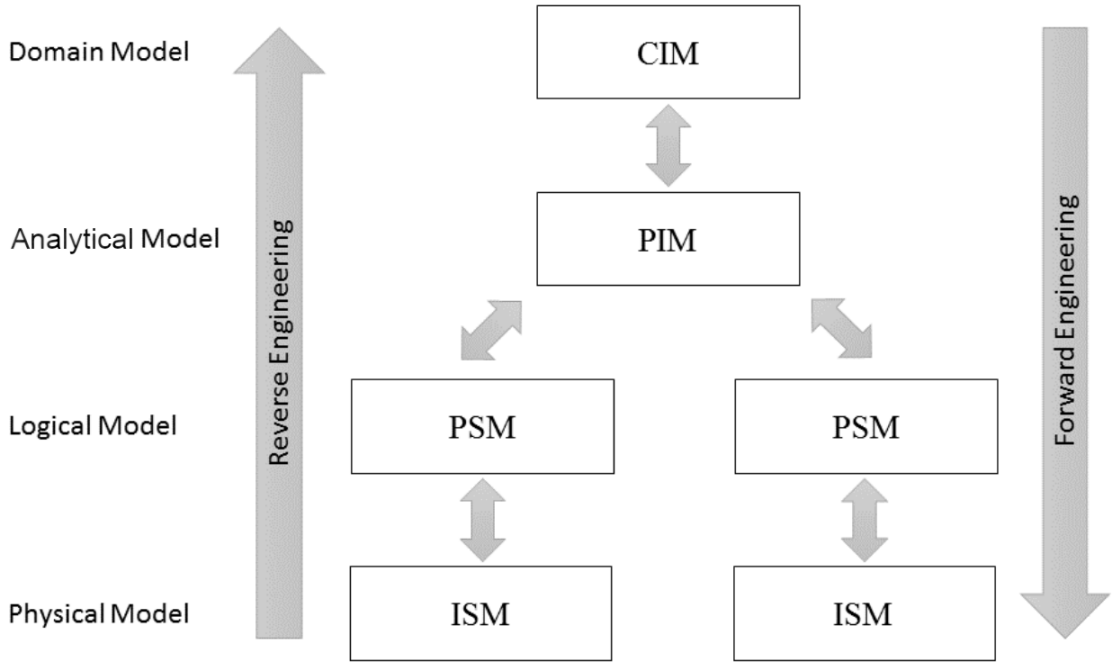


Figure 5.1: Models of MDA [117]

A similar approach towards more fine-grained concepts transition from the conceptual level to an implementation is exercised in BORM (Figure 5.2[3]).

An interesting solution is exhibited by the Normalised Systems Expanders (Chapter 6). While originally not focused on modelling, the NS Descriptors represent textual models in the end[4] and a code is being generated from them. The semantic gap is overcome in this case by generating NS Elements, higher-level components forming together a coherent class of applications — client-server CRUD applications — and also by introducing carefully designed customisations mechanism of the generated code.

5.1.1.1 Our Contribution and Applications

The author started to tackle this topic in 2006 together with Marek Pícka; In [193], we elaborated on the BORM method (Section 4.4) and the transition of its concepts, as the schema in Figure 5.2 had not been elaborated into an engineering method. The initial ideas were then elaborated together with Petr Šplíchal in 2011 [201], where we presented our achievements on method of transformations of the models between various phases of the engineering process. We studied BORM's models along with their meta-model. UML (Section 3.6.1) was selected as the output of the transformation.

[3]Here, the term "conceptual modeling" is also used in its confusing narrow sense.

[4]A new graphical NS modeller is currently being developed — Section 6.5

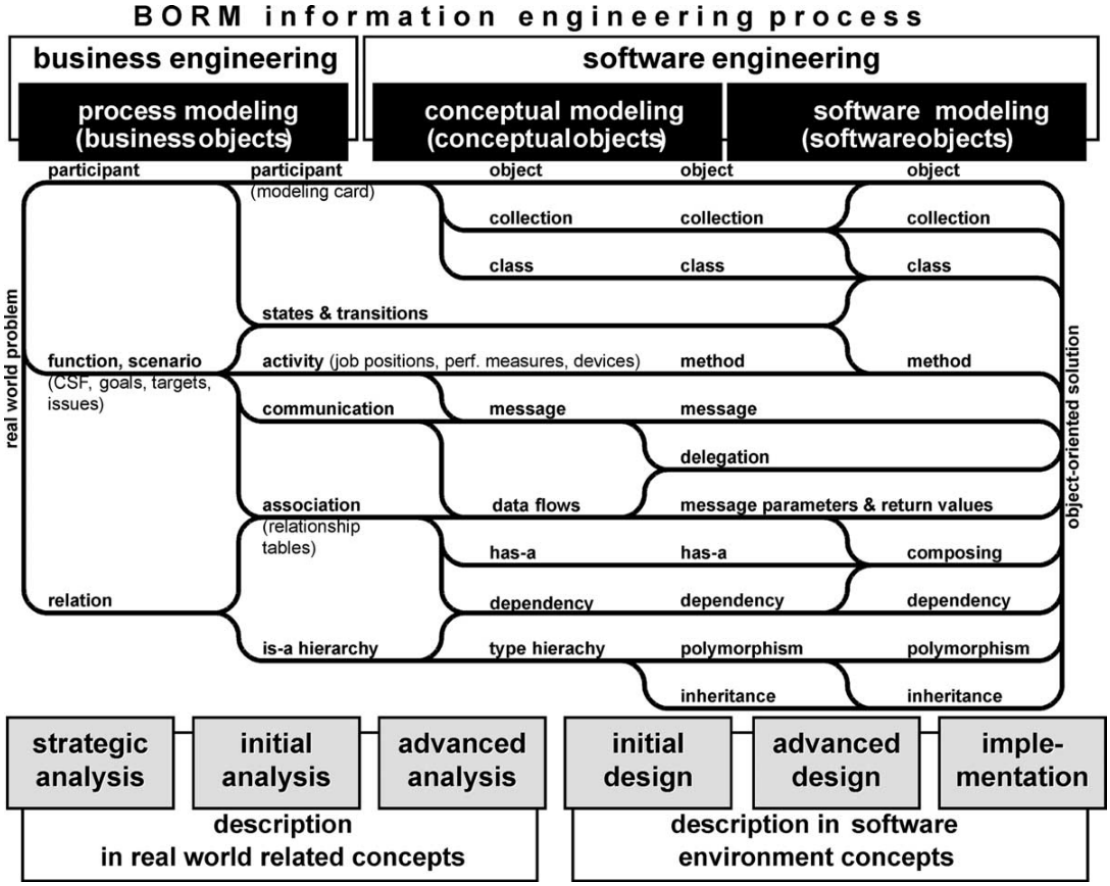


Figure 5.2: BORM: Evolution of concepts [95]

In the traditional MDA, UML models are typically used for PIM modelling. As explained in Section 3.6.1 and Section 3.4.1, OntoUML exhibits higher ontological conceptual modelling qualities. This is why we started a research on using OntoUML in MDE-based software engineering. Together with Zdeněk Rybola, we elaborated a method for transformation of OntoUML models into software-engineering models — UML in it's narrow sense (see Section 3.6.1).

The rationale of using OntoUML for MDD together with principles of transformation of OntoUML into UML were formulated by the author and introduced in [202] (Chapter 12). Zdeněk Rybola then continued with the author by elaborating a 2-step approach: (i) transformation of OntoUML into UML and (ii) transformation of UML into a relational database schema [203] consisting of transformation of rigid Sortal types in general [204], Kinds and Subkinds specifically [205] and anti-rigid Sortal types [206]. A comprehensive example can be found in [207]. The above-mentioned work is summarised in Rybola's dissertation thesis [117]. Next, elaborations about optimisation of the transformation were published in [208].

Additionaly, Rybola and Richta elaborated the third step, the transformation of the

PSM with the additional constraints into the ISM to define the constructs in the database to hold the data and maintain the constraints [118, 119, 118, 120, 121, 122, 117].

A contribution to this topic is also the bachelor's thesis [232], in which Matúš Vološin implemented generation of a Smalltalk code from an OntoUML model. Similarly, in a more detailed way, Dan Homola elaborated a transformation from OntoUML into a general object language and then into C# in his master's thesis [233].

5.1.2 Model Execution

In the end of Section 4.3.2.3 we mentioned that DEMO models can be directly executed, which is thanks to their high expressiveness. Kervel and ForMetis Consultants, GmBh, a Dutch company, developed DEMO Engine, an interpreter of DEMO models. The formal foundations were laid in [123] and the concept was successfully implemented in enterprise practice [90, 91, 124].

Another example of direct models execution is MEMO (Section 4.5). Its current implementation is based on XMF, a language execution engine featuring a metamodel called XCore [111]. Ulrich Frank explains: "XMF allows access to and modification of its own specification and its runtime system. Hence, there is no clear distinction between the language and a corresponding meta-language, and therefore XMF is reflective. Furthermore, it includes tools for building compilers for further languages. That makes XMF a meta-programming facility that allows execution of code written in different languages in the same runtime system." [102]

5.1.2.1 Our Contribution and Applications

The DEMO Engine originally supported the Construction and Process Models (Figure 4.6). Marek Skotnica pursued in his bachelor's [234] and master's theses [235] implementation of the Action Model rules.

We have been also applying enterprise engineering theories in software engineering in other ways. COPS GmBh is an Austrian-Czech software development company that develops complex financial software systems ("corima" being its flagship product). Ondřej Dvořák developed a confirmation engine according to ψ-theory [209] that has been implemented in the company.

While diving deeper into the topic of DEMO models executions, we realised a formal gap between the ontological level of DEMO models and an execution of them. While ontological models do not assume any realisation nor implementation (Section 4.3.2.2, Section 4.3.2.3), the execution of them materialises both. This can be related to a situation of a computation, where there is a mathematical formula and then a computer programme doing the computation. The gap is filled-in by a formal machine — e.g. the Turing Machine (Section 3.5). Similarly, we strived to fill-in the gap between the DEMO models and their execution with Marek Skotnica, resulting in the concept of the DEMO Machine [210], [211].

As industrial-grade BPMN/BPEL execution engines are available (see also Section 8.2), we elaborated a method for transformation of DEMO models into BPMN, which is de-

scribed in Section 4.3.4.1. Another work on transformation of DEMO models into an exectuable model is the master's thesis of Vejražková [61], who proposed a method of partial transformation of DEMO models into Petri Nets. Apart from that, her work contains a comparison of BORM, BPMN and DEMO.

Next, we also explored the other way — a tight integration of a software code and its conceptual model. Marek Suchánek in his master's thesis [236] developed a Haskell library enabling to express, visualise and verify conceptual models directly in the code, which represents a "bottom-up" approach.

5.2 Conceptualisation in Implementation

If we look back at the semiotic triangle (Figure 3.2), we can see that in case of software coding (being one of the key activities in implementation), the sign is the code. There is a two-way relation between implementation concepts and their encoding:

1. Creating a code as a process of expressing the (executable) conceptualisation.
2. Perceiving a code in order to create a conceptualisation in the mind (understanding the pragmatics of the code, Figure 3.4). We will speak about two situations:
   a) Interactive, live development
   b) Reverse engineering

While the first point has been elaborated in theory and practice of programming very intensively in the form of programming languages development and support tooling for code generation, the second point seems to be somehow neglected, especially the interactive development. Maintaining a strong conceptual and cognitive bound with the code has been a tradition of pure object-oriented technologies (Section 7.2) and also in functional programming (Section 7.3).

5.2.1 Interactive Development

A person very active in the area of interactive code perception is Bret Victor, an author of suggestive and mind-opening talks [125, 126, 127, 128], where he presents deficiencies, principles and also several solutions contributing to the problem (Figure 5.3). His principles inspired authors of interactive programming tools such as LightTable or ProtoREPL [129]. In his talk "The Future of Programming" [130], Victor also shows that there had appeared very progressive approaches and technologies in the past decades that got somehow lost in the history[5]. A case of one such "almost lost" technology is the Smalltalk programming language and its unique environment (Section 7.2). We describe some of the interactive tools in Smalltalk in Section 7.2.1.

[5]Because history is written by money and marketing, not by excellence (author).

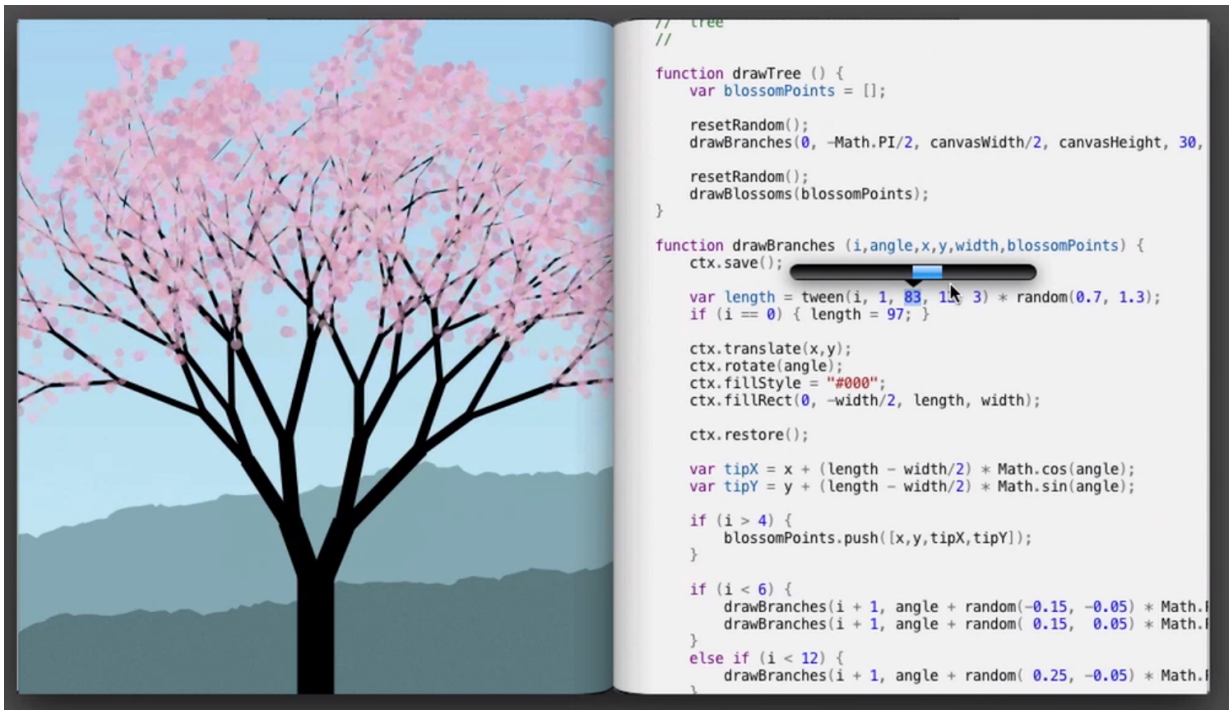


Figure 5.3: A dynamic correspondence between the code and its effect [126]

5.2.2 Reverse Engineering

Reverse engineering is "the process of analyzing the subject systems to

- Identify the system's components and their interrelationships and
- Create the representations of the systems in another form or at higher level of abstraction." [131]

There are several approaches possible [132, 133]:

Static Analysis based on structural information of the code.

Dynamic analysis uses execution trace of the programme.

Hybrid — the combination of static and dynamic.

An example of a static analysis approach is MOOSE, an extensible language-independent environment for reengineering object-oriented systems with the following characteristics: [134]

- It supports reengineering of applications developed in different object-oriented languages, as its core model is language independent which, if needed, can be customized to incorporate language specific features.

- It is extensible. New entities such as measurements or special-purpose relationships can be added to the environment.
- It supports reengineering by providing facilities for analysing and storing multiple models, for refactoring and by providing support for analysis methods such as metrics and the inference of properties of source code entities.
- Its implementation being fully object-oriented, MOOSE provides a complete description of the meta-model entities in terms of objects that are easily parametrised and/or extended.

MOOSE is built entirely in the Pharo programming environment (Section 7.2.1). It uses the Roassal visualisation library [135], which enables dynamic visualisations of the analysed code (Figure 5.4).

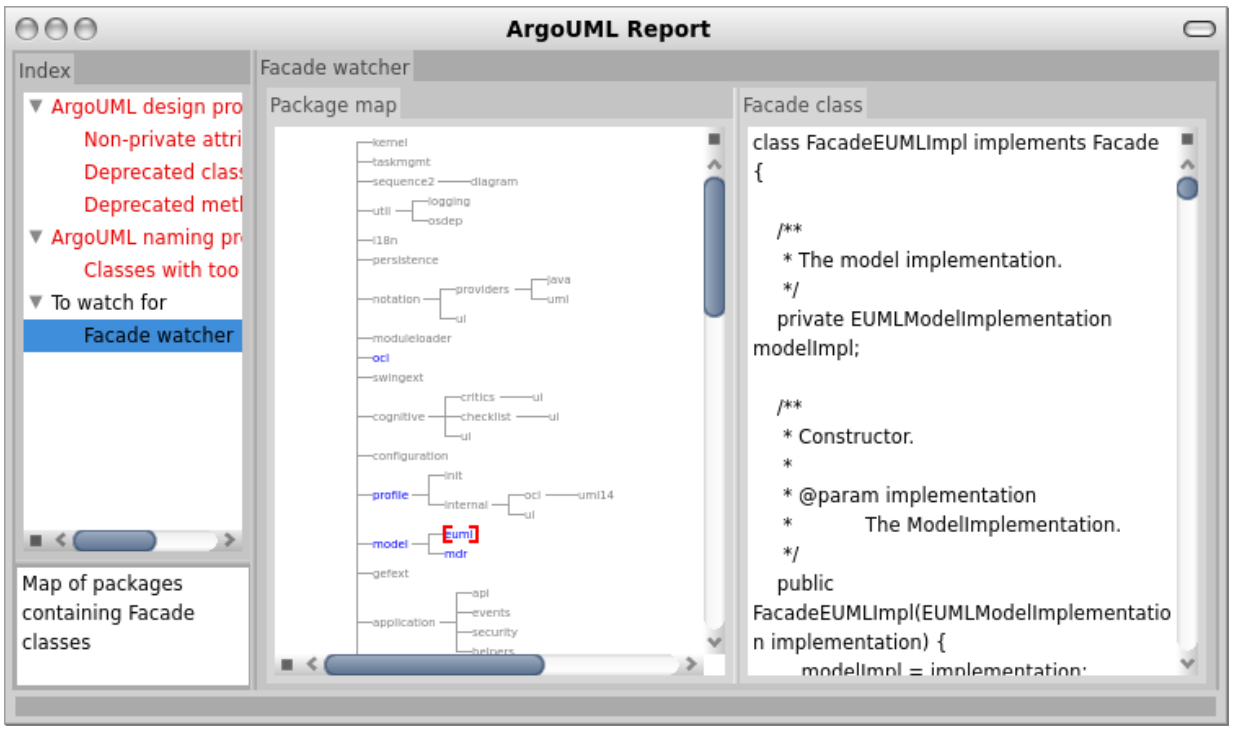


Figure 5.4: An example of a static code analysis in MOOSE [136]

5.2.3 Our Contribution and Applications

Veronika Larionova performed an extensive review of reverse engineering approaches and tools in her master's thesis [237] that is actually an exploration of possibilities for rejuvenation of applications by expanding Normalised Systems based on legacy code (see Chapter 6). This challenging topic will be also pursued in PhD research of Jan Bliznicenko.

Peter Uhnák developed a dynamic analysis tool for the Pharo programming environment [212]. Leveraging Pharo MetaLinks, it enables an unobtrusive analysis of the running code and generates instance diagrams (Figure 5.5). The motivation for developing the tool was helping to understand complex code, in this case `Spec`, Pharo's graphical user interface library.

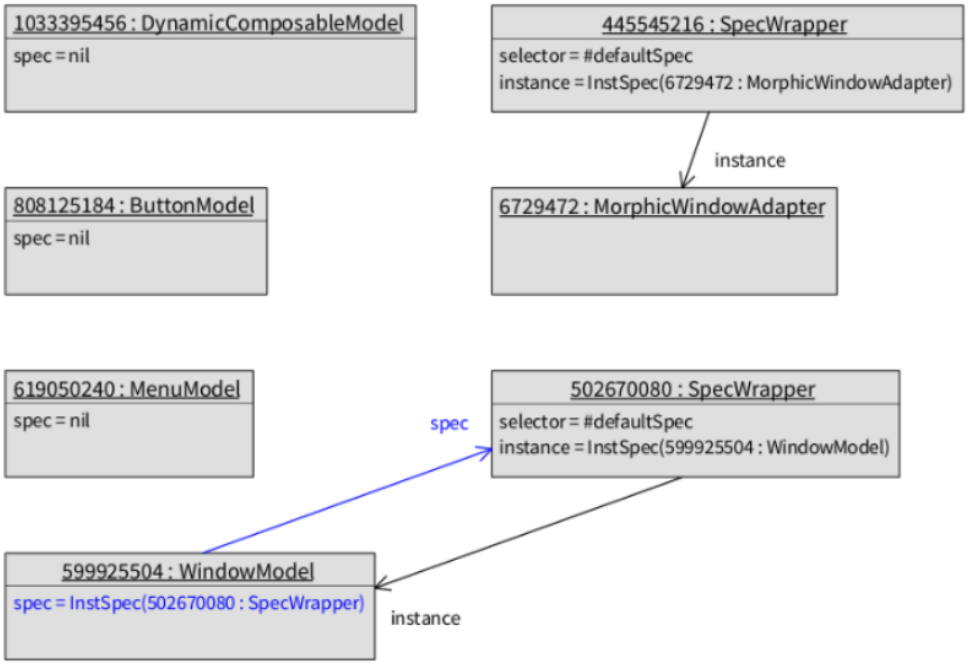


Figure 5.5: An Instance Diagram with an unexpected link from SpecWrapper to WindowModel

Chapter 6

Evolvability and Normalized Systems

6.1 Introduction

In the first decades of the 21st century, organizations are operating in hypercompetitive environments, constantly monitoring their environment for new business opportunities and striving for customer satisfaction by delivering products and services of unprecedented quality. This has resulted in enterprises that are faced with challenges such as increasing complexity and increasing change. These challenges require high evolvability, agility or flexibility of the organization and its information systems. However, current methodologies for the development of information systems struggle to meet this demand for flexibility [137, 138][1].

6.2 Principles

The theoretical framework of Normalized Systems is based on *systems stability theory* and *entropy* [139, 140]. It applies an analysis of the *modular structure* of software architectures of information systems. This modular structure consists of constructs in programming languages such as procedures, functions, classes, services and, most recently, aspects. The framework consists firstly of 4 principles, which indicate when so-called combinatorial effects occur in a modular structure. A **combinatorial effect** exists when the size of the impact of a change to a software architecture is dependent on the size of the information system [137, 138]. Combinatorial effects therefore represent a particularly harmful kind of coupling in the software architecture, as such effects cause information systems to become increasingly difficult to maintain during their life cycle, until they cannot be maintained any more in a cost-effective way and ultimately have to be replaced by an information system with similar functionality. In other words, combinatorial effects explain why current

[1]Most of the text in Section 6.1 to Section 6.4 was used from unofficial materials provided by Jan Verelst, with his kind permission.

software architectures are inherently limited in their flexibility. Furthermore, they also have highly a negative effect on other quality factors such as reusability.

6.3 Elements

The goal is therefore to build information systems free from combinatorial effects. To that end, the theoretical framework contains 5 elements, with which the basic functionality of virtually all information systems can be built. These elements are [139]:

- a data element for storing data,
- an action element for the execution of a calculation or algorithm,
- a workflow element for the execution of sequences of action elements,
- a connector element for input and output functionality, and
- a trigger element for time- or status-based execution of action elements.

An application, then, can be built consisting of N instances of these elements. These instances are parametrised copies of the 5 elements, and can therefore be built using a kind of code generation, which is called "element expansion". As it can be proven that these elements do not contain combinatorial effects, it is also guaranteed that the resulting application is free from combinatorial effects, and therefore guaranteed to be more evolvable and reusable than current information systems [139, 141].

The Normalized Systems theoretical framework applies to any system consisting of modular structures and is therefore completely independent of specific programming languages and the use of specific packages or frameworks. Hence, the elements can in principle be built in any combination of technologies (including programming languages, packages and frameworks). The essence of the framework remains that certain "errors against evolvability", i.e. combinatorial effects, systematically need to be removed from modular structures which results in "evolvable modularity", irrespective of the particular language or framework being used.

6.4 Applications and Research

The practical relevance of NS has been demonstrated by NSX, a spin-off company of the University in Antwerp that built so-called **NS Expanders** — a system that generates normalized client-server web applications with the structure described above. The code is generated (expanded) from **NS Descriptors** containing the structural and behavioural specification. Specific functionality currently not possible to express in specifications is then hard-coded as so-called **customisations**. The Expanders have been already applied in tens of mission-critical complex business software systems.

Additionally, a research on Normalized Systems is performed at the department of Management Information Systems of the University of Antwerp. This research aims at, for example, advancing the theoretical framework at the software architecture level, as well as the application of the Normalized Systems principles to business processes and organizations [142, 143, 144, 145, 146]. The goal is to strive for "evolvable modular" organisations, which can satisfy the ever growing demand for flexibility.

6.5 Our Contribution and Applications

We started cooperation with the University in Antwerp in 2012. The first achievement became the bachelor's thesis of Kolařík [238], who explored the idea of applying OntoUML (Section 3.4.1) for conceptual modelling of Normalized Systems. The promising approach took some time to settle and it is now going to be further pursued in the PhD study of Marek Suchánek.

Next, Janeček explored in his master's thesis options for alternative database backends for the NS Expanders [268].

Currently, there are several joint research topics and projects we pursue together with the University in Antwerp:

1. PhD research of Ondřej Dvořák focused on component software systems design (Section 6.5.1)
2. PhD research of Marek Suchánek focused on conceptual modelling of precise technical systems (Section 6.5.2)
3. Evolvability of documents (Section 6.5.3)
4. PhD research of Vojtěch Knaisl focused on applying functional paradigm and functional programming languages for expanding of server-side normalized web applications and PhD research of Jan Slifka focused on applying functional paradigm and functional programming languages for expanding of client-side normalized web applications (Section 6.5.4)
5. Building NS Conceptual modeller by Peter Uhnák (Section 6.5.5)

Let us now briefly describe these projects.

6.5.1 Component Software Systems Design

In the PhD research by Dvořák, we try to get nearer McIllroy' vision of software assembling instead of programming it. He dreamed about components organized into standardised libraries:

"... I would like to see components become a dignified branch of software engineering. I would like to see standard catalogues of routines, classified by precision, robustness,

time-space performance, size limits, and binding time of parameters. I would like to apply routines in the catalogue to any one of a large class of often quite different machines, without too much pain. . . " [147]

We apply enterprise engineering theories, namely τ-theory and β-theory (Section 4.2) to achieve a controlled engineering approach for managing the relationship between system function and its construction (F/C), thus facilitating changing technology challenges more rigorously and efficiently. We formulated the notion of *Affordance-Driven Assembling (ADA)* and its simplified version Objectified *Affordance-Driven Assembling (O-ADA)*, which together with the so-called *Semantic Descriptions* represent a software-engineering approach enabling reasoning about users and their purposes versus components and their properties. Our experiments show that engineering methods based on these theories may increase reusability of code and improve important metrics such as costs, time reduction and error rate decrease, especially when switching to a new technology. [213]

We approach this goal by exploring the relationship between F/C. In particular, we apply a software design approach to build component-based systems by clearly defining the function, construction, and F/C relationship of components. After the first formalisation steps and prototypes, we now work on a domain-specific language for semantic descriptions, independent on a specific programming language, as well as on further evaluation in practice.

As component-based software development poses also "Increase flexibility, decrease usability" trade-off influencing the effectiveness of reusing artefacts [148], we addressed this fundamental topic in our research. In [214], we explain that equally flexible components can considerably differ in usability costs. Therefore, the architecture of components matters to evaluate final cost on building software. We proposed a model of building components that can help to decrease costs on software development, while providing a demanded level of flexibility.

6.5.2 Conceptual Modelling of Precise Technical Systems

This research embodies the ultimate vision of CCMi depicted in Figure 3.1, because its ambition is to connect the "ultimate ends" — a pure (i.e. with no technical details) business conceptual model up to the technical specification.

The first exploratory work was done by Marek Suchánek in his master's thesis [236], in which we went "bottom-up" by creating domain annotations in code (namely Haskell) thus enabling business domain modelling being tightly bound to code.

In our collaboration with the University of Antwerp, we now pursue the "top-down" approach, i.e. designing a set of interconnected models from the domain-based down to the implementation-based, similarly to MDE (Section 5.1.1), but encompassing not only a structure, but also a behaviour. The first exploration of this approach was the mentioned master's thesis of Kolařík [238]. The topic will be now further pursued by Marek Suchánek in his double-degree PhD research.

6.5.3 Evolvability of Documents

This side project builds on the preliminary work of Oorts [149], who applied Normalized Systems theories in the domain of document management, namely managing a university curriculum — study program designs exhibit by their nature large amounts of dependencies due to constraints of prerequisite courses, courses being taught in several study programs, etc. These characteristics make managing and changing study programs very complex, on occasion even preventing study program changes. In his work, he brought solutions to these challenges based on the concept of modular and evolvable system design. Basic engineering concepts such as modularity, coupling and cohesion were used to explain and illustrate the evolvability and traceability of study programs.

Our motivation is to generalise Oorts' approach and design a technical solution for managing course knowledge in an evolvable manner. We first focused on making a proper domain ontology of modular documents [215]. Our next step is to design a prototype solution for managing the OntoUML.org portal contents [150].

6.5.4 Applying the Functional Paradigm for the Normalized Systems Expanders

Functional programming and functional languages are traditional approaches with known benefits (Section 7.3). Today, we witness a renaissance of them — new languages and solutions for building high-quality enterprise software systems are emerging and practically every modern programming language offers at least some support for functional programming. Their superiority to the imperative and object-oriented programming in terms of testability, parallelization, reusability and provability along with high expressiveness are the motivation for exploring what they can bring with combination of NS. The hypothesis is they support NS principles in a better way that other programming approaches. As such, the expanded code may exhibit better qualities itself and with the respect to customisations.

The PhD research of Knaisl that starts in September 2018 is focused on exploring the functional paradigm for typical server-side tasks like data processing, data exchange with other systems, CRUD, suitable models of persistence, etc.

The parallel PhD research of Slifka (also starting September 2018) addresses the challenges of the client side: interactive user interface, DOM manipulations traditionally complicated by incompatibility of different web browsers and precipitous development of web technologies — browsers, standards and especially libraries and frameworks, again with respect to code qualities and customisation, again with respect to code qualities and customisations.

6.5.5 Normalized Systems Conceptual Modeller

The first version of NS Descriptors format was a textual technical language [139] that was not easily accessible for non-programmers. Because of that, the Prime Radiant was

developed, being a web-application (a NS system itself) for more user-friendly creation of NS Descriptors [151]. While being more user-friendly, the Prime Radiant is effectively a graphical user interface for NS Descriptors, thus still being a challenge for less technical analysts.

The NS Conceptual Modeller is a joint project of the University of Antwerp, NSX and CCMi focused on developing a full-featured CASE tool providing visual diagrams, verifications and generation of NS Descriptors. The solution is based on our open conceptual modelling platform, OpenPonk (Section 8.3.2). Most of the current work was done by Peter Uhnák in his master's thesis [239] and he continues his work in NSX now.

Chapter 7

Software Implementation

Enterprise engineering sees implementation as a general means of "giving life" to the designed conceptual model — there is a multitude of possible implementations spanning from pure human involvement via primitive technical systems up to modern IT. Figure 3.1 depicts some of the implementation concepts. We will now elaborate on "materialisation" of conceptual models in the form of software artefacts.

7.1 The Discipline of Software Engineering

Before we start speaking about software implementation, we should root it in the discipline of software engineering.

The IEEE authority defines software engineering as *"the systematic application of scientific and technological knowledge, methods, and experience to the design,* ***implementation****, testing, and documentation of software"* [152].

This definition names the most important activities and phases of which we deal with implementation and it also expresses our ultimate goal: systematic application of scientific and technological knowledge and methods, in our case through rigorous application of conceptual modelling (Section 3.3) in the context of enterprise engineering (Chapter 4).

What does it mean to look at software implementation from the perspective of conceptual modelling? In our approach, it means looking at the code literally as at a conceptual model. Technically, programme code is a textual model[1] specifying a conceptualisation of structures and computation [188]:

- For the imperative programming paradigm, the key concept is *instruction*.
- For the structured (imperative) programming paradigm, the key concepts are also *sequence*, *selection* and *iteration*.

[1]There are even approaches of graphical programming languages, such as Aardappel, LabVIEW or GRAPNEL, however, they have not achieved much popularity.

- In the modular programming, the concept of *module* is introduced.
- In the object-oriented paradigm we speak about *objects*, *messages*, *attributes*, *classes*, etc.
- For the case of functional paradigm, *function* and their *composition* are key concepts.
- In the logical programming paradigm, *predicates* are the key concept.

From the conceptual point, software development means formulating terms and their relations in the selected conceptualisation (programming paradigm). We abstract here from topics such as time and space complexity, technical limits and failures and other low-level concerns. Instead, we focus on qualities such as *expressiveness*, *evolvability* (Chapter 6), *understandability*, *soundness*, *coherence* and *concinnity*.

Let us now briefly introduce the paradigms we are dealing with in our research from this point.

7.2 The Object-Oriented Paradigm

The object-oriented (OO) paradigm originated in 1960s for programming simulations (the Simula programming language). It influenced many today's object-oriented languages, however in its pure form it was mainly developed in XEROX Parc in 1970s and 1980s by the team of Alan Kay, who, being a biologist, based OO on the conceptual model of living organisms — a notion of an autonomous **object** with some internal structure and behaviour communicating with the outer environment through **message sending** [153] (Figure 7.1). Today, the object-oriented paradigm is blurred by a lot of hybrid concepts causing confusion, however, this is the true essence of OO, as meant by Alan Kay, who confirmed this e.g. in 2003: "OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things." [154]. This conceptualisation enables to abstract out "technicalities"[2], which is why pure OO is one of the corner stones of implementation explorations in CCMi.

While there are tens of hybrid object-oriented languages (i.e. imperative languages with object features), languages that can be considered pure are few – Smalltalk [155] and Eiffel [156] being the most well-known representatives, while Smalltalk was developed in XEROX Parc along with the paradigm and technical implementation. Thanks to its purity, the language is very simple, the syntax fits literally on a post card (Figure 7.2).

It is probably not by chance that the community around pure object-oriented technologies has traditionally paid a lot of attention to conceptual and cognitive aspects of programming (Section 5.2). Throughout the decades, the **live coding** principles and tools have been developed and improved. The first Smalltalk implementation in XEROX Parc was already supporting live coding, debugging and inspecting objects on-the-fly [153]. Successive implementations such as Visual Age for Smalltalk by IBM or VisualWorks Smalltalk

[2]Which come from legacy, demands for high performance, systems integration and other aspects.

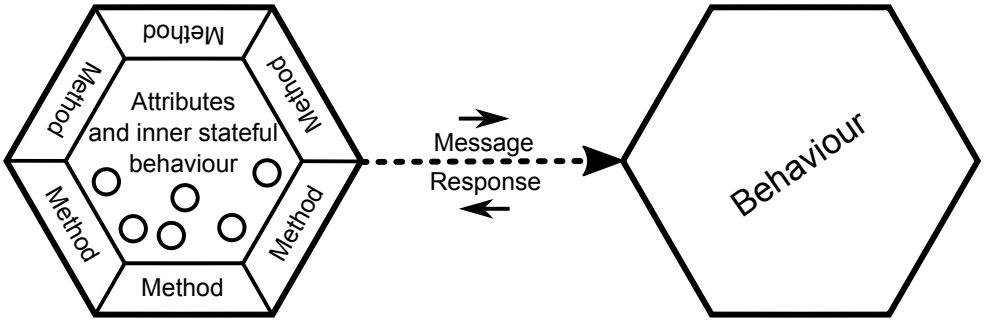


Figure 7.1: The object-oriented paradigm by Kay (inspired by [153])

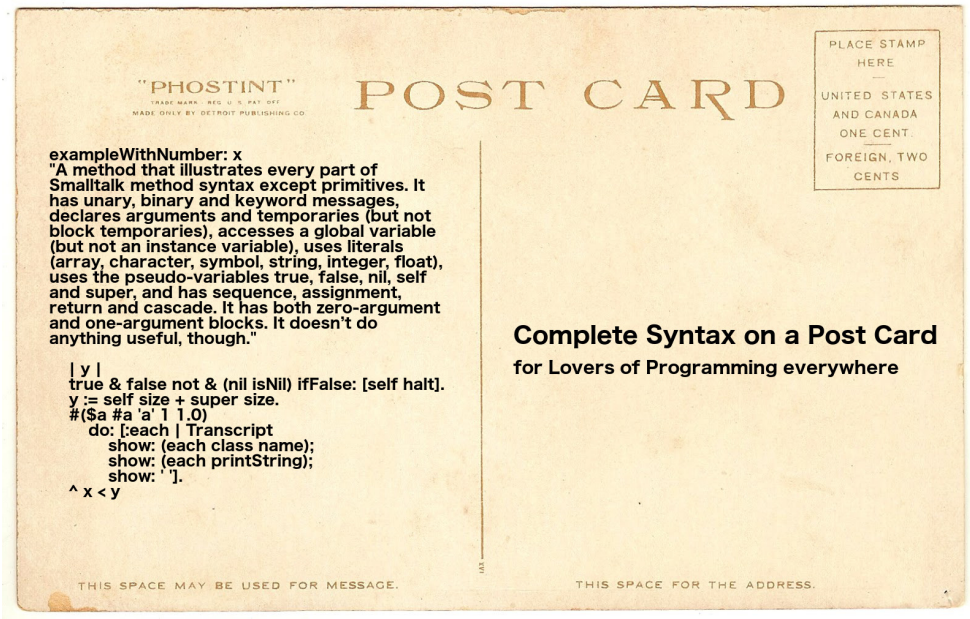


Figure 7.2: The syntax of Smalltalk fits on a post card [157]

pushed these possibilities further. Visual Works is today the major commercial Smalltalk implementation, while Pharo stands for the major open-source implementation.

7.2.1 Pharo

Pharo is a pure object-oriented programming language and a powerful environment, focused on simplicity and immediate feedback [158], thus enabling interactive live development (see Section 5.2.1). The "Glamorous Toolkit" developed under the MOOSE project (Section 5.2.2) provides "moldable" tools that have been integrated into the Pharo base:

Moldable Inspector further extends possibilities of traditional Smalltalk object Inspector by providing intelligent views based on content [159, 160]. Moreover, extending the

inspector by custom views is possible (Figure 7.3).

Moldable Spotter enables quick search of concepts in the system, similarly to Spotter on Mac OS. It can be also easily extended to search user-defined objects. Using an internal DSL, the authors have created 100 custom searches for 30 different data types. On average, extending Moldable Spotter with a new type of search required 8 lines of code [161].

Moldable Debugger — The traditional Smalltalk debugger enables live interaction with the system, i.e. debugging on-the-fly, often without restarting the running code. The Moldable Debugger goes further by enabling developers to create specific debuggers with custom debugging operations for stepping through the execution and custom user interfaces [162], Figure 7.4.

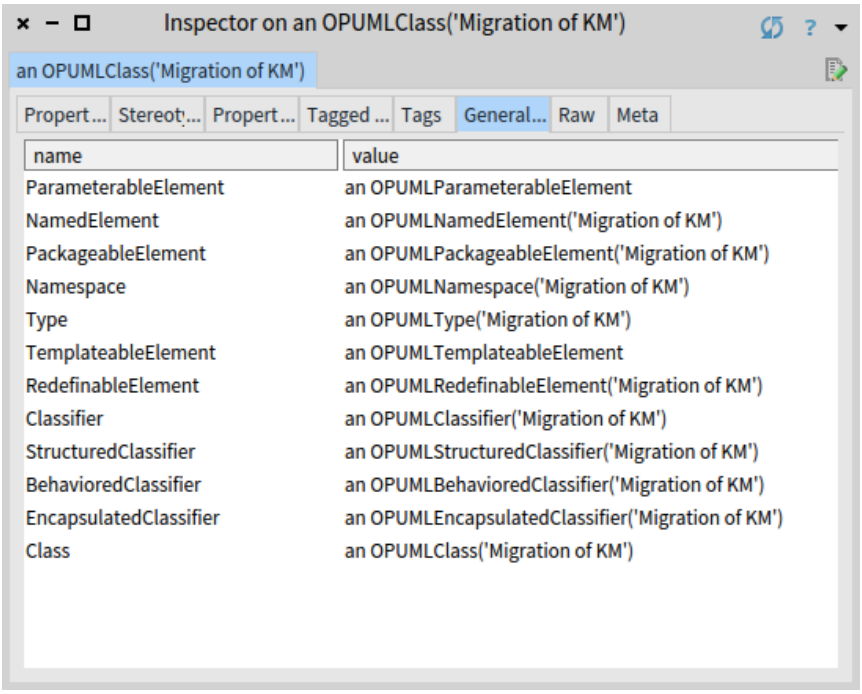


Figure 7.3: Moldable Inspector with several views (screenshot from OpenPonk, Section 8.3.2). The Generalisation view is custom.

7.2.2 Our Contribution and Applications

We have been utilising Pharo in teaching of pure object-oriented paradigm (BI-OMO, BI-OOP) and also in projects:

- Vulnerability assessment tool — protection of cultural heritage against floods by the author [216]

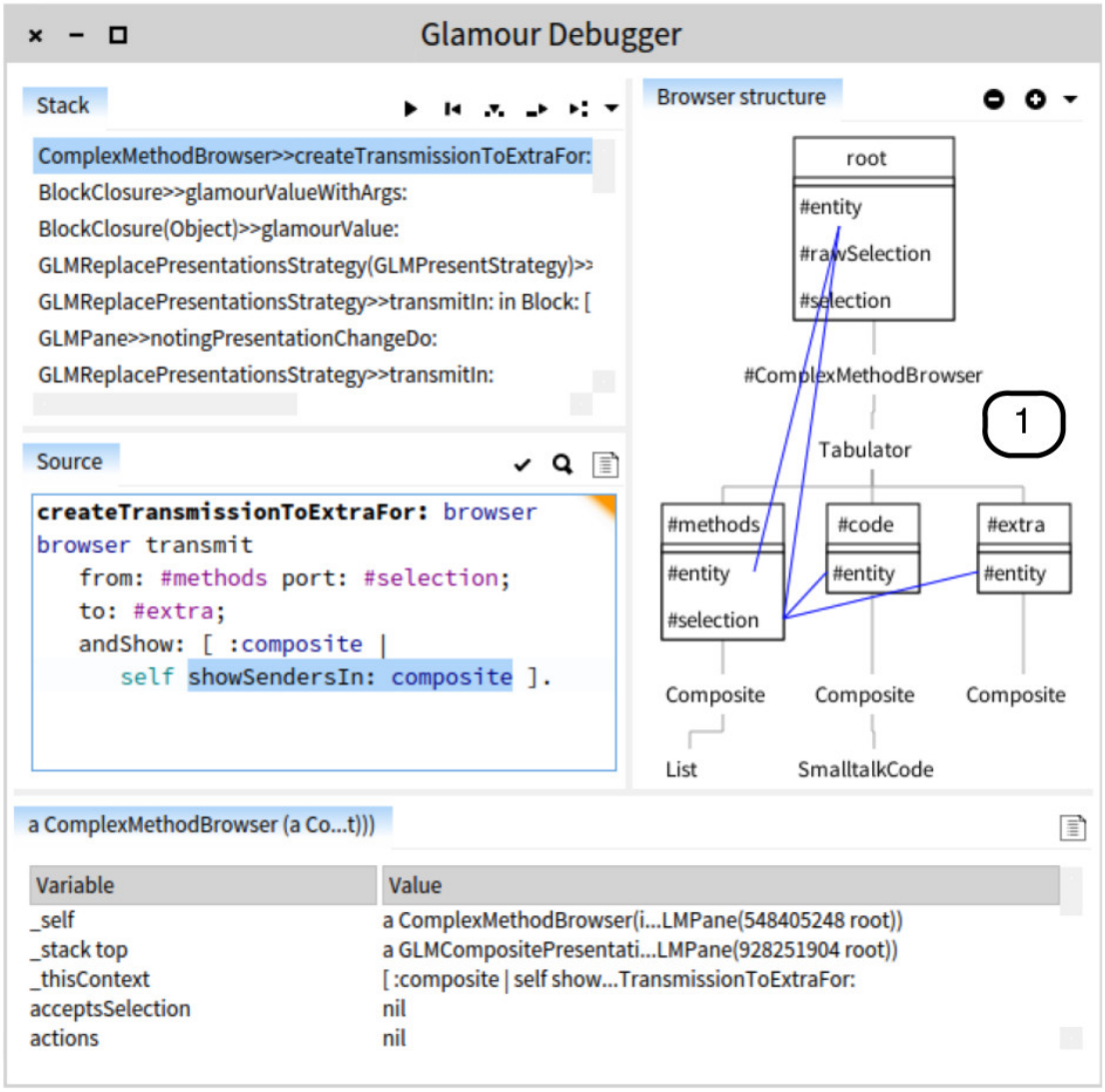


Figure 7.4: A domain-specific debugger for Glamour: (1) visualisation showing the model of the browser currently constructed [162].

- DayWork.cz — a short-term job portal; the backend was developed in Pharo by Michal Balda [240] and then continued by Jan Blizničenko.
- Live visualisation of epidemiological models — implementation of modular models for the Kendrick epidemiological modeller backend [241], [217]
- OpenPonk — an open platform for conceptual modelling (Section 8.3.2)
- Runtime Object Structure Visualisations (Section 5.2.3)

We are active members of *Pharo Consortium* — a body of legal entities, who use and contribute to Pharo [163]. Exceptionally active has been CCMi member Peter Uhnák, the

chief programmer of OpenPonk (Section 8.3.2), who contributed to the Roassal visualisation library used to provide the diagramming support and the Spec UI framework. Apart from more than 10 versatile libraries, his contributions include code contributions, debugging, detailed bug reports, extensive discussions, and providing help to new and existing developers alike (Pharo mailing list (2000+ mails), StackOverflow (top 5% contributor to UML, top 10% to Smalltalk and Pharo)) [239].

7.3 The Functional Paradigm

"Functional programming (FP) is so called because a program consists entirely of functions. ... Typically the main function is defined in terms of other functions, which in turn are defined in terms of still more functions, until at the bottom level the functions are language primitives." [164]

"The special characteristics and advantages of functional programming are often summed up more or less as follows. Functional programs contain no assignment statements, so variables, once given a value, never change. More generally, functional programs contain no side-effects of any kind. A function call can have no effect other than to compute its result. This eliminates a major source of bugs, and also makes the order of execution irrelevant — since no side-effect can change the value of an expression, it can be evaluated at any time. This relieves the programmer of the burden of prescribing the flow of control (and enables a compiler/interpreter to do advanced optimisations — author). Since expressions can be evaluated at any time, one can freely replace variables by their values and vice versa — that is, programs are 'referentially transparent'. This freedom helps make functional programs more tractable mathematically than their conventional counterparts." [164]

In these two paragraphs, Hughes explains the essence of the functional paradigm, another programming approach focused on simplicity, conceptual purity and high practical relevance.

7.3.1 Theoretical Foundations

As the the main principles of functional paradigm are discussed in [218] (Chapter 19), we will not introduce them here.

7.3.2 Practical Relevance

Twenty five years after writing the fundamental paper "Why Functional Programming Matters" [164], Hughes returns together with Hu and Wang to the topic and they evaluate "How Functional Programming Mattered" [165] and they enthusiastically state: "Functional programming is now at the forefront of a new generation of programming technologies, and enjoying increasing popularity and influence". Observing the further rise of functional programming languages and solutions, we can say that the trend gets even stronger in the recent 3 years.

In several respects, the purely-functional system surpasses an object-oriented system, as is explained in [219] (Chapter 18). On the other hand, key challenges of the functional paradigm are:

- *Side effects*, such as disk reading and writing and user interactions.
- Modelling an *order* of operations.
- Modelling *time* in the system.

While these are exactly the strong points of OOP (Section 7.2), there are also approaches in FP addressing them:

1. Allowing side-effects in the system, thus allowing impure parts. This is the way of Lisp or Clojure [166, 167, 168].
2. Using advanced functional abstractions — the Haskell programming language uses various types of *Monads* that simulate impure behaviour in a pure way [169, 170, 171].
3. Introducing *infinite streams* through lazy evaluation and the abstractions of *functional reactive programming* (see below).

7.3.3 Interactive Development

The functional programming practice typically also gives focus to live interactive development to maintain the concept-sign correspondence (Section 5.2.1). This happens through REPL (Read-Eval-Print-Loop) tools [172] available in FP implementations. They are traditionally command-line interactive tools, however modern REPLs can be built in the programming environment (e.g. LightTable) and they can even support graphical output (ProtoREPL [129], Figure 7.5).

At the same time, currently, pure object-oriented technologies seem to offer more immersive interactive development experience[3].

7.3.4 Challenges of FP in Software Engineering

Traditionally, functional programming had been a domain of computer science. LISP actually stands for "List processing" and this was a typical task apart from scientific computations and artificial intelligence. However, along with IT development, the FP languages implementations matured into general-purpose industrial-strength solutions. Also, new modern implementations started to occur (such as Clojure or Rust) and practically all traditional high-level programming languages offer pure functional features[4]. A great warp

[3] Although the author does not have any research results on this topic at hand.

[4] e.g. Java since version 1.8 and C++ since version 11 have lambda functions, Groovy has a strong support for FP [242], etc.

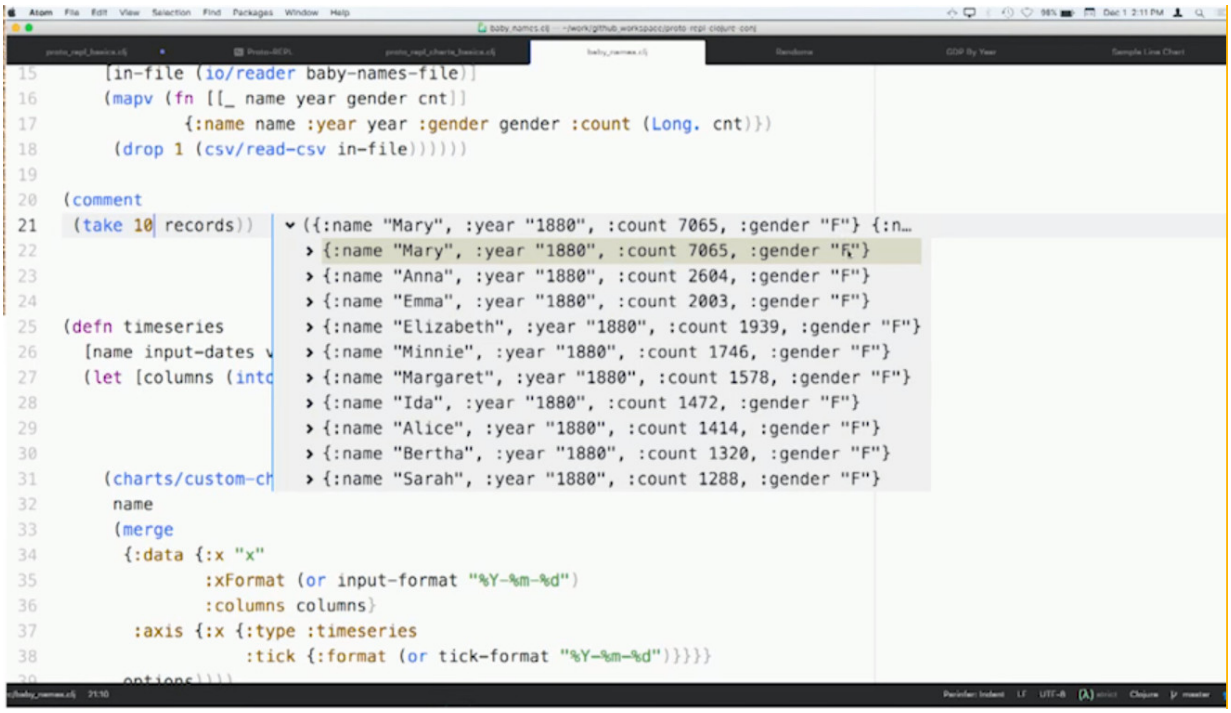


Figure 7.5: ProtoREPL live evaluation during coding [129]

for functional programming also happens in client-side web programming, where there is already a serious assortment of FP languages that compile to Javascript, not mentioning that JavaScript itself can be used as a purely-functional programming language [173], [218].

So, we have a paradigm with long history and experience and its modern implementations available. However, software engineering based on pure functional paradigm seems not to be in its full. We argue that there are two main reasons:

1. Switching to functional paradigm from traditional imperative-style thinking (including object-orientation) is not easy, as every major paradigm shift.

2. There seems to be a gap between the programming in small (which was the traditional ground of FP) and programming in big — assembling modern complex modular architectures.

7.3.5 Our Contribution and Applications

Functional paradigm due to its conceptual purity and benefits described above is one of the interesting topics of CCMi. Compared to other topics, this is the youngest one, so we have now more topics and opportunities than results. Our motivation comes from observing a gap that seems to be between the functional programming paradigm and modern software engineering, i.e. addressing point 2 from above. The important topics from our perspective are:

- *Model-driven Engineering* seems practically missing in the world of functional programming.
- *Missing higher-level functional design patterns* similar to [174]. While some patterns of the object-oriented paradigm are trivial thanks to first-class functions (such as the Command Pattern), there seems to be missing a generally-agreed cookbook on solving enterprise software systems architectural patterns in FP.
- *Dynamically vs statically-typed.* The "battle" between static and dynamic typing seems to rage over ages with no tangible results. There has been attempts to deal with this dichotomy such as [175, 176, 177], however sound engineering approaches to the topic still seem to be scarce.
- *Applications of the paradigm* to address modern challenges, such as graphical user interfaces [178], big data and databases [179], client-server web applications [180] and others have been undertaken, however, again, there seems to be missing engineering methodologies that would enable broader industrial adoption.

The relation between pure functional paradigm and domain conceptualisation, namely in the form of a DSL is shown in [219] (Chapter 18).

There were several bachelor's theses exploring the application of functional programming in practice, mostly for web application development [243], [244], [245] and user-interface generation [246].

The first systematic attempt to tackle the topics in CCMi was a PhD research of Lukáš Janeček. We started with putting together conceptualisation of the paradigm in [218] (Chapter 19) and showed their presence in a surprisingly traditional language: JavaScript. As, sadly, Lukáš abandoned his PhD studies, the line of research was suspended and it is now being resumed in two PhD research topics of Vojtěch Knaisl and Jan Slifka. The research will be performed in collaboration with the University of Antwerp and the Normalised Systems Expanders will be used as a practical relevance ground. The goal is to start conceiving the missing engineering methodologies for leveraging modern functional programming in enterprise engineering, both at the server-side (Vojtěch Knaisl) and client-side (Jan Slifka).

Apart from research involvement, we have been using pure FP in our projects. The most notable is *Data Stewardship Wizard* [220], [247], [248], a joint project of ELIXIR-CZ and ELIXIR-NL, where tooling for data stewards and researchers to help with devising data management plans is being developed. We develop the software using pure FP technologies — GHC (Haskell) on server side and Elm on client side.

Similar technologies have been used also in the *Marrow Donor Registry Simulator* project (Section 3.5.1).

Chapter 8

Tooling

Modern engineering is unthinkable without proper software tools. Software and enterprise engineering are no exceptions. We deal with two types of software engineering tools here:

- Computer-aided engineering tools (CAE)
- Execution tools

8.1 Computer-Aided Engineering Tools

Computer-aided engineering tools (CAE) is the broad usage of computer software to aid in engineering analysis tasks. In our discipline, these are CASE tools (Computer-Aided Software Engineering) (see Section 5.1.1) and CABE (Computer-Aided Business Engineering).

Although the name "CABE" is nicely expressive and coherent, it is used rarely[1] and the reader can rather find them under other names such as *Business Process Modelling Tools*, *Enterprise Modelling Tools* or *Business Process Management Tools*. The aim of CABE tools is similar to CASE — to facilitate creating a model of an organisation (typically in the forms of diagrams), to help it visualise, communicate, verify and generate implementation or at least implementation models. Examples of such tools are Sparx Systems' Enterprise Architect [181] or Visual Paradigm [182]. Some CABE tools are even integrated with execution engines, so the models may be directly executed (the next section).

8.2 Execution Tools

8.2.1 Business Process Management Tools

While implementation of structural concepts leads to database systems (see e.g. Section 5.1.1.1), execution of behaviour is a more complex topic, both in the theoretical

[1]The author found it mentioned in a work of a renowned Czech professor Václav Řepa, however just in Czech, so no reference is provided here.

level (Section 3.5) and the practical level. While database systems are arguably present as an information support in all modern enterprises and there is a plethora of their types, providers and licenses, behaviour-supporting systems are considerably more scarce. In case of enterprise engineering, we talk about process execution in the context of **Business Process Management (BPM)**.

The terminology is not clear in this area, however Jaroslav Fibichr did a nice job in his master's thesis [269] to describe the state-of-the-art of this topic and he summarises that "BPM is defined as a set of principles, methods and tools used to identify, design, execute, monitor and control business processes". The life-cycle is depicted in Figure 8.1.

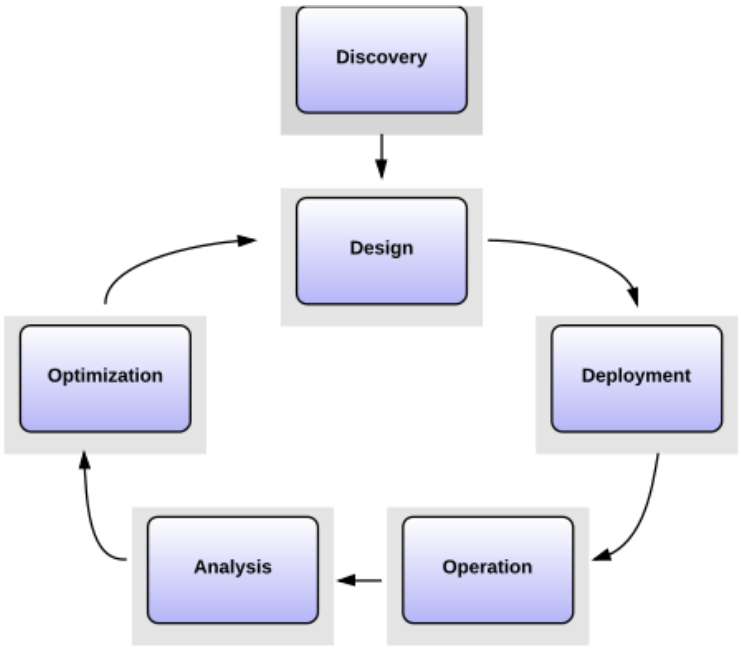


Figure 8.1: Business Process Management life cycle according to Smith and Fingar [183]

While BPM does not necessarily implicate software support (its history starts sometime in 18th century with Adam Smith), we are interested in tools here: "**Business Process Management System/Suite (BPMS)** refers to a set of (software) tools that support the continuous process improvement efforts in the organization across all phases of the process life cycle." An overview of current biggest players in the field is given by Figure 8.2.

We can see that the definition of BPMS suggests a certain overlap with CABE and also other tools out-of-scope of this text, such as decision-support tools. However, we are interested specifically in the *Operation* phase, as it is where business process execution takes place.

Source: Gartner (October 2017)

Figure 8.2: Business Process Management Suites magic quadrant

8.2.2 Workflow Management Systems

Workflow management systems can be seen as a subset of BPMS — a workflow system "organizes the routing of case data amongst the human resources and through application programs" [184]. Although workflow management has obviously also its roots surpassing software engineering, today's workflow management systems began to occur in mid 1990s[2] by "take the business processes out of the applications" approach [184].

[2]Their origins can be traced to 1970s to XEROX Parc and the "Office Automation Systems" project.

First workflow management systems were modelled using Petri Nets, the current industrial systems use typically BPMN (Figure 8.3, Section 3.6.2).

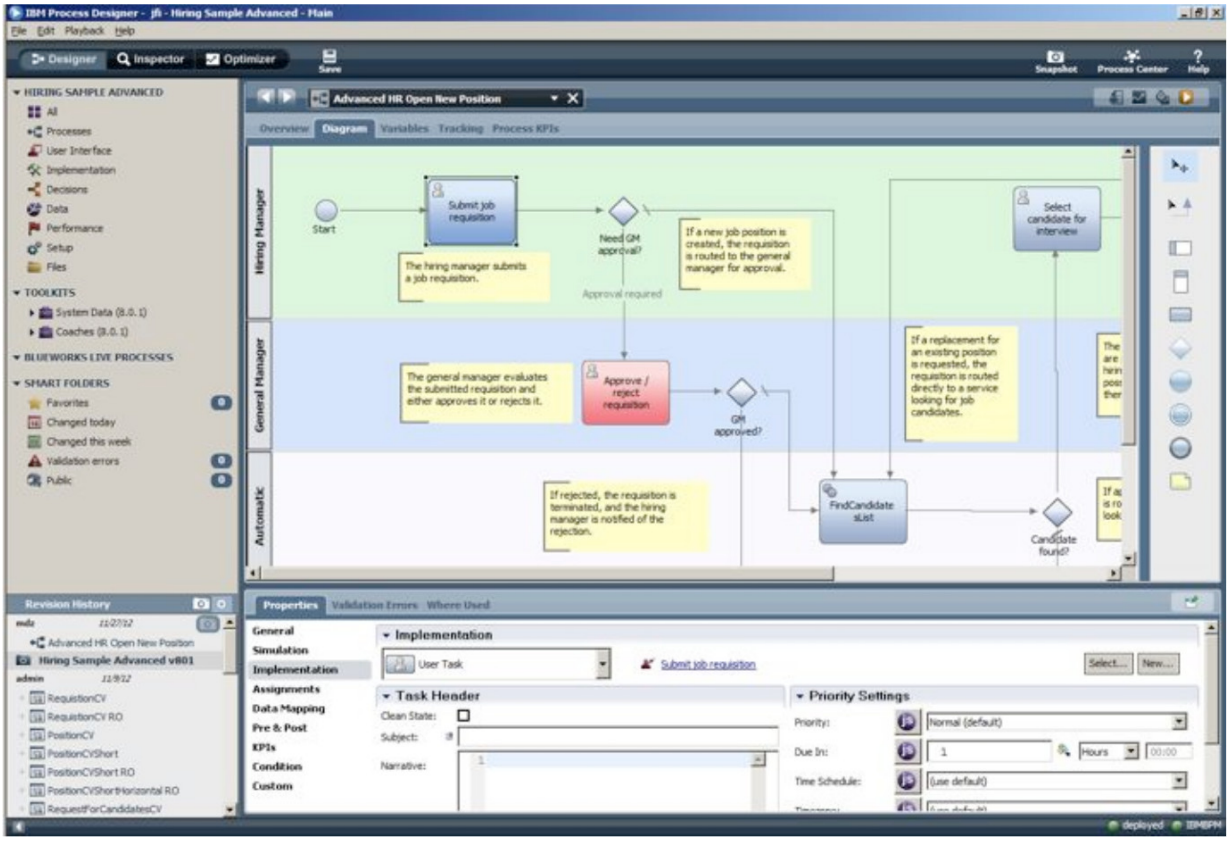


Figure 8.3: IBM Process Designer

A good example of a workflow execution engine based on the model-driven architecture is IBM BPM (Figure 8.3), where there are typically several layers of process models, the highest level being the business process model and the lowest being "boxes with code" implementing the low-level technical tasks, such as reading data from a database. However, the transitions between models are still quite coarse and limited by the BPMN notation expressiveness and its possibilities of modularisation (see also Section 3.5). Moreover, the solution is proprietary and high licence fees limit adoption by budget-tight smaller companies.

The situation in open-source workflow engines is even further from an MDE ideal — the gap between the business and technical level is quite broad and deep: usually a single level of BPMN models is accompanied by a loosely-coupled code.

8.2.3 Our Contribution and Applications

Joint projects of the Faculty of Information Technology and the Centre of Knowledge Management at the Faculty of Electrical Engineering towards building university work-

flow systems created a considerable experience with both commercial (IBM BPM) and open-source systems, which is elaborated in bachelor's theses of Klára Jelínková [270] and Vladimír Šimon [271]. Roman Lanský then presented achievements in creating modular components in IBM BPM in his bachelor's thesis [272].

Next, in Section 4.4.4, we explained our work on executability of BORM models. We continued our theoretical work into the form of process simulation tool that started as a student team project and was then continued in bachelor's theses by Michal Balda [249] (using the pure OO paradigm) and Oskar Maxa [273] (using the functional paradigm). Alžběta Zyková then designed and implemented a true workflow engine prototype in her bachelor's thesis [250]. This workflow engine was planned to be used in the project "Protecting Cultural Heritage from Floods" funded by the Ministry of Culture of Czech Republic [216], unfortunately, it has not happened (from reasons unrelated to the tool).

As for DEMO, we explained our work done on execution of DEMO models in Section 5.1.2.1. Moreover, Roman Lanský in his master's thesis [251] designed and developed a prototype of a messaging and task-management application based on the ψ-theory and Radek Buša designed and implemented a prototype of a web-based modeller of DEMO diagrams that generates web forms in his bachelor's thesis [274].

8.3 A Conceptual Modelling Platform

Research in the field of conceptual modelling that has an ambition to be not merely theoretical comes at certain point to the question of how to implement the ideas of new notations and algorithms over models such as verification, inference, models transformation, code generation, etc. In CCMi, we had come to this point as well[3].

There are generally three typical ways how to approach this task:

1. Programming a tool "from the scratch". This has a benefit of greatest freedom and flexibility, however, implementing a diagramming software properly is not a trivial task and given limited resources, such academic projects usually lead to user-unfriendly tools incompatible with the rest of the world, doomed to being abandoned and forgotten — sadly with the elaborately implemented results of its authors.

2. Using a meta-modelling tool [185] such as MetaEDIT+. This class of tools enables (at a user level) implementing own modelling notations and algorithms. However, the user is limited by the abilities of the tool and its extensibility has inherent bounds compared to the first approach. Moreover, the author has not found a usable tool with an open-source license.

3. Using a conceptual modelling platform, which has a form of loosely-coupled libraries and frameworks available in a general-purpose programming language.

[3] In fact, this topic had been in the mind of the author even before the CCMi conceiving.

8.3.1 Our Contribution and Applications: OpenCASE/OpenCABE

We took the third way. Podloucký started developing an open platform for conceptual modelling in 2010 using the Eclipse Modelling Framework (EMF) for the Java platform. The result in 2012 was a tool for the BORM method, which implemented some of the author's ideas, such as a distinction between entities and elements ([221], Chapter 10), business properties and open API ([222], Chapter 20). The name of the tool started as OpenCASE, which was later changed to OpenCABE along with stronger orientation towards business engineering. One of the strong points of the tool is also generating of non-technical-people-friendly operational manuals ([223], Chapter 21).

8.3.2 Our Contribution and Applications: DynaCASE/OpenPonk

The OpenCASE/OpenCABE project was success in its ability to deliver a usable CASE tool for the BORM method that was used in projects, as well as in teaching, while materialising research results. However, it failed in delivering an accessible open platform for conceptual modelling, as the complexity of learning EMF and OpenCASE's architecture showed to be tremendous, thus effectively preventing, for example, students to elaborate bachelor and master theses using the platform. The second weak point was low interactivity with models in the modelling and meta-modelling process in the spirit of Section 5.2.1.

The author hypothesised that these failures resulted from rigid static nature of the Java platform. This is how the idea emerged to implement a conceptual modelling platform in Pharo (Section 7.2.1). The hypothesis was that dynamic interactive nature of a pure OO language implementation will enable to significantly reduce code boilerplate and increase dynamic interactive behaviour of the tool. The project started as an exploration by a student team project. The most active team member, Peter Uhnák then continued in development. The achievements are summarised in his bachelor's thesis [252] and [224] (Chapter 22). The hopes laid in this project by the author were fulfilled — the foundations of an open platform for conceptual modelling had been laid in a few semesters and the code base is a fraction to the OpenCASE's. The tool enables live interaction with models and prototyping of algorithms, visualisations and simulations.

The greatest excitement showed to be the learnability of the platform — already several students were able to master the platform and implement solutions in their theses, such as verification of models [253], reports generation [254], simulation [255] or implementing new notations and modelling tools, such as DEMO [256], BPMN [257], OntoUML instance models [258] and OCL interpreter [226].

This project also undertook a name change — the original name "DynaCASE" was expressing the dynamic nature of the platform. However, there was an inconvenient name clash with an existing project and the name also did not reveal the ambition to be actually a plaform for tools development. In the end, the OpenPonk name was chosen, "ponk" being a Czech word for a "workbench".

This versatile workbench has been used in two major projects. The first was facilitating the design of agent-based models and a code generation to promote participatory

modelling [186] — a project where round-trip engineering was implemented to generate code from models.

The second project is the NS Conceptual Modeller introduced in Section 6.5.5.

Chapter 9

Final Thoughts

Here ends the introductory part and the reader is definitely eager to emerge into reading the chapters. However, there are some notes that are due to be written here.

First of all, it was quite a challenge to order the text in a flow, as the presented topics are highly related and intertwined. I hope that the ordering "middle-left-right" from Figure 3.1 gave a good logical flow of the text. At the same time, before emerging into the chapters, the reader may like to glance back at Table 2.1 to refresh the topics categorisation in its full.

Next, I would like to share an observation about conceptual modelling. It is a discipline in some respects similar to taiji or yoga: there seems to be a cloud of myth and esoterics that may even repel so-called "practical people", however, once you emerge in them, you discover that they are merely about realising, training and applying simple principles that permeate everyday life. Conceptualisation is the same — practitioners of all engineering disciplines do them — often unconsciously — every day and they do not realise how much it defines their job and its outputs. A master of conceptual modelling is similar to a yoga master: they are able to master trivial tasks much better and they gain almost "supernatural" abilities for managing hard tasks; in case of conceptual modelling these are abilities to master complexity, assist domain experts in formulating precise specifications, analysts and designers in proper ways of transforming them into a sound design and developers of all kinds in their realisation. Unfortunately, compared to yoga and taiji, the society and practitioners still do not realise the potential of this discipline fully. My wish is that a reader after reading this work can sense a glimpse of this potential.

The above is also related to a relative youth of the discipline of conceptual modelling. While being here literally since antiquity, ontology and semiotics had been rather a philosophical interest without much practical relevance. This changed tremendously when a focus of human development moved from dealing with material things into dealing with immaterial concepts (Figure 3.2). Today, the society lives mostly in an artificial conceptual space consisting of social norms, commitments and claims on one hand and artefacts designed to embody these concepts on the other — enterprise and software engineering being probably the most representative examples of disciplines revolving almost exclusively

around this new artificial human world.

The youth of the "modern conceptual modelling" inevitably means lack and immaturity of methods and tools. Looking from the positive side, there is a wide space of research and engineering topics to be addressed. In this work, I showed the topics tackled by CCMi in our research and projects, however the problem space is so huge that working on one topic opens an explosion of new research topics. We try to balance the breadth/width ratio and we are positive that in the coming years, we will go both deeper in existing topics and have more capacity to possibly address additional ones, as was already sketched in the text.

Last but not least, there is a warning. The text may create a nice vision of lego pieces falling together into right places thanks to having solid formal foundations, engineering methods and tools. Indeed, "There is nothing so practical as a good theory", as Kurt Lewin suggests, however, there are (sadly) limits, as Goethe counters: "All theory, dear friend, is gray, but the golden tree of life springs ever green." We can observe these two poles, for example, in a harmoniously complementary cooperation of our fathers of enterprise engineering: Jan Dietz and Jan Hoogervorst. While Jan Dietz tamed the design of enterprises in the Lewin's way and gave us solid formal foundations and sound DEMO methodology, Jan Hoogervorst pinpoints the gravity of the social level of enterprises that bring disturbances into formally designed models[1].

Similar discussions take place in software engineering, which has been turning from rigid software project management to agile approaches emphasising communication, feedback and other humane aspects. Some authors go so far to relate software development more to growing plants than an engineering discipline. You can, maybe surprisingly, find exactly this attitude in the legendary "The Pragmatic Programmer" [187].

This having said, I strongly believe that studying formal approaches, formulating engineering methodologies and growing supporting tools is a strong way forward both in enterprise and software engineering. It is for a great good that "soft" disciplines are here to remind us about our human nature and what is really important in life, as did Karel Čapek, a famous Czech novelist, who gave the concept of a "robot" to our technical world.

[1]Referring here to the brilliant keynote of Jan Hoogervorst "The Imerative for the Empoyee-Centric Theory of Organizatin and its Significances for Enterprise Engineering" given at EEWC 2017 in Antwerp.

Part II

Chapters

Chapter 10

Supporting Enterprise IS Modelling using Ontological Analysis

[221] Pergl, R. Supporting enterprise IS modelling using ontological analysis. *Lecture Notes in Business Information Processing*, volume 88 LNBIP, 2011: pp. 130–144.

Supporting Enterprise IS Modelling using Ontological Analysis

Robert Pergl

Department of Information Engineering,
Faculty of Economics and Management,
Czech University of Life Sciences,
Prague, Czech Republic
pergl@pef.czu.cz

Abstract. The goal of this contribution is to show that incorporating ontological analysis into modelling of enterprise information and knowledge systems during the software engineering process may bring considerable benefits. Necessary terms related to ontological analysis are defined, the most important being the Concept Map of the Domain (CMoD). The BORM method is then analysed from the ontological point of view and the ontological analysis is showed in the context of the layered Model Driven Architecture approach. Possible gaps that may occur between the layers in practice are studied. Two methods how to incorporate the ontological analysis into the modelling process to overcome those gaps are presented and discussed.

Key words: ontologies, ontological analysis, enterprise systems modelling, BORM method, Model Driven Architecture

1 Goal

The goal of this contribution is to show that incorporating ontological analysis into modelling of enterprise information and knowledge systems during the software engineering process may bring considerable benefits. The concept is shown using the BORM method. Theoretical foundations and methodological aspects are provided as well.

2 Methodology

Necessary terms related to ontological analysis are defined, the most important being the Concept Map of the Domain (CMoD). The BORM method is then analysed from the ontological point of view and the ontological analysis is showed in the context of the layered Model Driven Architecture approach. Possible gaps that may occur between the layers in praxis are showed, and their consequences are discussed. Two methods how to incorporate the ontological analysis into the modelling process to overcome those gaps are presented and

their additional benefits together with drawbacks are discussed. The more complex of the methods is presented on a practical example. In the end some related work is mentioned and conclusions are future work are formulated.

3 Motivation

Modelling is a crucial part of the software engineering process. Its goal is to create a set of models that in the end result in technical specification of the system and enable a successful implementation by technical experts (database architects, programmers, designers, etc.) that typically have just a vague knowledge about the problem domain. For the modelling to be the most effective and efficient, *consistency* of elements in the models must be maintained. This may be a hard task, because often we need to put the same elements into various diagrams of the same type and there are elements in different diagram types that are closely related to each other. Changes made to diagrams during the model evolution thus may provide very painful tasks of consistency maintaining.

Consistency of diagrams is crucial for *requirements traceability*. It ensures and documents that the life cycle is being followed [4], because

- It demonstrates the relation between the modelling inputs and outputs.
- It ensures that every model is based on some predecessor that has its foundation in some requirement.
- It contributes to model validation.

Consistency and requirements traceability offer these additional benefits [20]:

- Certification support
- Impact analysis
- Maintenance
- Project traceability
- Changes to older systems
- Reusability
- Risks mitigation
- Testability

Moreover, as put in [15], pure software-engineering conceptual modelling may be short in many areas, ontology modelling among many others.

We also felt somehow motivated by the statement published in [6]:

> *Without ontologies, or the conceptualizations that underlie knowledge, there cannot be a vocabulary for representing knowledge. Thus, the first step in devising an effective knowledge-representation system, and vocabulary, is to perform an effective ontological analysis of the field, or domain. Weak analyses lead to incoherent knowledge bases.*

4 Definition of Terms

4.1 Ontology

The word **ontology** comes from the Greek *ontos* ("being") and *logos* ("word"). It generally studies the *categories of things* that exist or may exist in some domain [18].

Ontologies are used in a wide range of scientific and engineering applications and thus there are several definitions that stress various facets of ontologies. We will present a few that are closely related to our goal:

Definition 1. ***Ontology*** *is a specification of a conceptualization. [11]*

This definition stresses the modelling nature of ontologies: *Conceptualization* means an abstract, simplified view of the world consisting of concepts, objects and other entities that are assumed to exist in an area of interest, and the relationships that exist among them. *Specification* means a formal and declarative representation.

Definition 2. ***Ontology*** *is a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic. [12]*

This definition includes the term *vocabulary* which we will utilise for referring to the terms in a subject area.

Definition 3. ***Ontology*** *is the* basic structure *or* armature *around which a knowledge base can be built. [19]*

For us, ontology will be a basic structure or armature around which a *model* of an information or knowledge system can be built.

4.2 Concept map

Definition 4. ***Semantic network*** *(or* ***concept map****) is knowledge representation technique that attempts to reflect cognition [5].*

Concept map is one way how to express ontology [9]. We will use concept maps as a convenient and visual way of ontology description. Let us introduce it formally:

Definition 5. ***Concept map*** *is a directed graph G_{CM}, where*

- *V_{CM} is a set of* vertices *that represent terms of the domain. Each vertex is denoted by the corresponding term identifier.*
- *E_{CM} is a set of* edges *that represent relations between the terms. Each edge is denoted by the relation identifier.*
- *M_{CM} is a mapping that assigns an ordered pair of vertices (v_1, v_2), $v_1, v_2 \in V_{CM}$ to every edge $e \in E_{CM}$.*

Similarly to the Model Driven Architecture [8], we specify several layers of ontologies:

Definition 6. ***Concept map of domain*** D ***(CMoD)*** *is a concept map, where vertices represent terms from domain* D *and edges represent relations between the terms.*

Definition 7. ***Concept map of domain model*** DM ***(CMoDM)*** *is a concept map, where vertices represent elements of domain model* DM *and edges represent relations between the elements.*

Definition 8. ***Concept map of metamodel*** MM ***(CMoMM)*** *is a concept map, where vertices represent elements of metamodel* MM *and edges represent relations between the elements.*

According to the "industry standard" for information and knowledge systems modelling, the UML notation [7], we may distinguish *structure diagrams* and *behaviour diagrams*, which represent two main views of a model. We will stick to this categorisation, however we selected the BORM method [13] for our purpose, because it encompasses business modelling and has a strong emphasis on conceptual correctness and consistent gradual transformation of terms [14].

For our purpose we will consider the ontology of a model in the BORM method as depicted in Figure 1. This is not a complete ontology, just a simplified subset that contains concepts and relations crucial for our needs.

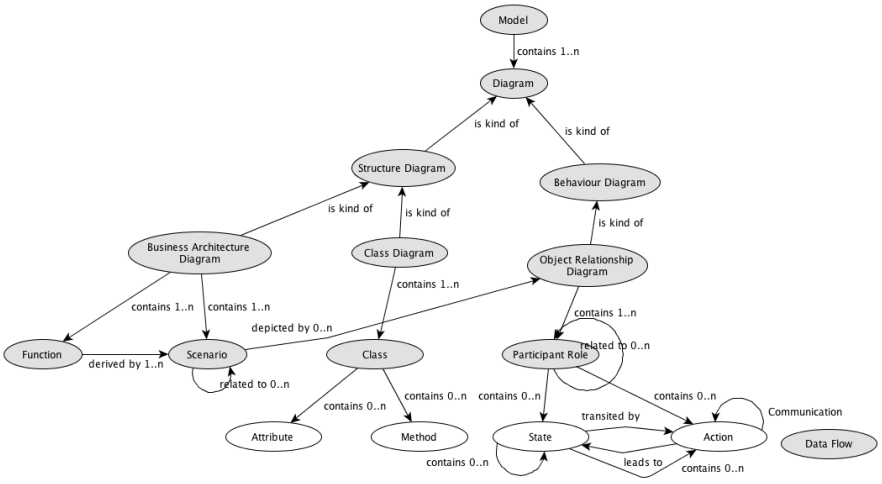


Fig. 1. Concept map of a BORM model (partial)

Data Flow as a concept can not be consistently drawn into the presented concept map. Formally, Communication is a relation between two activities whose

meaning is "Activity X communicates to activity Y". This relation has two attributes: *Input Data Flows* and *Output Data Flows*, each being a set of Data Flows (Figure 2). Unfortunately this cannot be expressed using just the graph theory formalism[1].

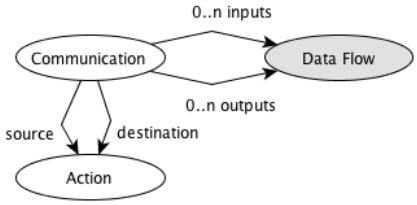


Fig. 2. Concept map of Communication

The whole architecture of models including ontologies is depicted in Figure 3[2]. Lower-level elements are *linguistic instances* [3] of higher-level elements.

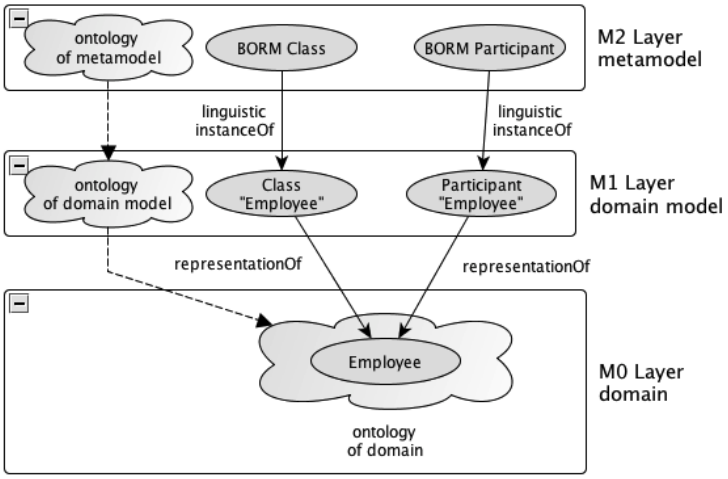


Fig. 3. Layered architecture of ontologies

This model maintains consistency in each layer and consistency between the layers. However, in real-life modelling we usually need to put more elements representing the same model term – e.g. `Employee` may be a participant in both

[1] Formally this would lead for instance to two mappings from the set of Communications into the set of Data Flows.

[2] For our purpose, we do not need the fourth, meta-metamodel layer.

the scenarios `Ordering` and `Invoicing`. The situation is depicted in Figure 4. Various elements of the same modelling term require to keep track of the *representationOf* relations in the model to maintain the identity:

Definition 9. *We say that the model element x is* ***ontologically equivalent*** *to the model element y if and only if there exist relations $representationOf(x,t)$ and $representationOf(y,t)$, where t is an element of the CMoD. We denote this $x \overset{\circ}{=} y$.*

The absence of *representationOf* relations in the model then leads to **element identity loss**, which leads to inconsistencies in models. When for instance the user wants to rename `Employee` to `Company Employee` and renames just one representation of it. Element identity loss also paralyses reportings and simulations. Unfortunately, most CASE tools do not support maintaining of *representationOf* relations. There is partial support e.g. in the Craft.CASE tool (`www.craftcase.com`), but it is not complete[3].

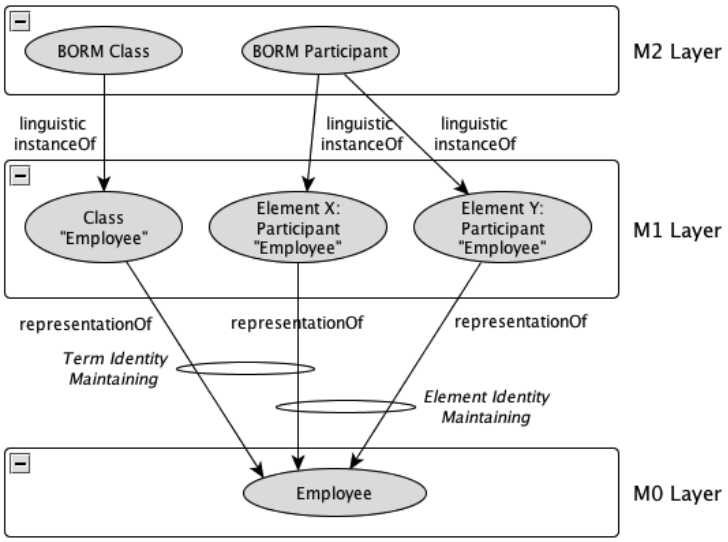


Fig. 4. Ontology maintaining the identities

The consistency between layers M1 and M2 makes no problem in practice, since every model element is instantiated from its class when put to model, and CASE tools track this relation. However, the consistency between layers M0 and M1 is usually practically omitted during modelling. The author does not know any CASE tool that would e.g. keep track that the class `Employee` and the participant `Employee` are two representations of the domain term `Employee`.

[3] More about this will follow later.

Without the explicit concept map of a domain and due to omitting the *represenationOf* relations between the layers M0 and M1, the **term identity loss** occurs. This may result in inconsistencies of the terms in the M1 layer. On the other hand, maintaining the identity of terms in the M1 layer through the *represenationOf* relations from the M0 layer enables sophisticated consistency checks, reporting and change management.

4.3 Semantics of the Concept Map

As mentioned in 3, an important role of ontology is to specify a vocabulary of a **domain**, specifically a **controlled vocabulary** that provides a finite list of terms together with an unambiguous interpretation of those terms. Every use of a **term** from a controlled vocabulary then denotes *exactly the same thing* in the domain.

While the M1 and M2 layers are implied by the used methodology (here BORM), the M0 layer is a subject to domain conceptualizations and, consequently, domain ontologies are established by the consensus of the domain experts. The terms used to express these domain ontologies must be rooted in a domain independent philosophically and cognitively well-founded system of real-world categories, i.e. a foundational (upper-level) ontology [10].

In ontological analysis, we may distinguish two categories of terms according to the modelling context:

Terms with the domain context – Those terms are uniquely identified by the context of the domain. For the BORM method those are:
- Business Architecture Diagram
  - Function
  - Scenario
- Class Diagram
  - Class
- OBA Diagram
  - Participant Role
  - Data Flow

"Being uniquely identified by the context of the domain" means that for example the class `Employee` or the scenario `Issue and order` are unique terms in the domain of `Trading company X`.

Terms with the model context – As opposed to the first category, these terms are uniquely identified in their context only. In the BORM method those are:
- *Attribute*, *Method* – identified in the context of the Class. Thus the `name` of a `Company` is a different term than the `name` of an `Employee`.
- *State*, *Action* – identified in the context of the Participant Role.
- *Communication* – identified by the pair (Source Action, Destination Action)

We use dot-notation for denoting the context[4]: `Company.name`, `Employee.name`, `Employee.waits`, `Employee.sends` and we call such term a **fully qualified term**.

Labels of activities and states may consist of several words and even several sentences. In such case we may enclose the terms in quotes, e.g. `Employee."sends the invoice"`, which is mandatory if the term label contains dots. Quotes in the label must be escaped using the backslash character, which is a well-known mechanism from programming languages[5].

BORM OBA diagrams permit to make nested process inside a participant state[6]. In this situation the context of nested states and activities would be the full context from the participant to the state/activity, e.g. `Employee."prepares an order"."collects requirements".consults`

5 CMoD: the Method

We will present two possible methods for creating and maintaing a Concept Map of a Domain.

5.1 Vocabulary Method

This is a method that brings the least possible additional activities to the modelling process while offering the benefit of identity maintaining. This approach is aligned with the Agile Modelling [2] in a sense that the ontological analysis is performed just in the most necessary extent. The method is based on *on-demand* introduction of terms into the concept map. It may be summarized into several points:

1. Start with an empty CMoD.
2. Perform the usual modelling in the M1 layer and elaborate the necessary diagrams and their elements.
3. If there occurs a need to reuse an element e in some other context (diagram), i.e. to create the element e' that is a copy of element e:
   a) Insert an element t into the CMoD representing some domain term.
   b) Create the relation $representationOf(e, t)$.
   c) Create the relation $representationOf(e', t)$.
4. If there occurs a situation where you create a new element y of class u in the layer M1 and there exists an element x of class v in the same layer such that u is ontologically equivalent to v:
   a) Insert an element t into the CMoD representing some domain term.

[4] This is just a matter of choice, another schemes like using slashes, arrows, etc. work the same.

[5] Again, this issue may be solved using various approaches.

[6] This is depicted in Figure 1 by the "contains" relation between two States and a State and an Activity.

b) Create the relation $representationOf(y, t)$.
c) Create the relation $representationOf(x, t)$.

An example of this situation may be `Employee` of class Participant and `Employee` of class Class. The names, however, may generally differ, although it is not recommended from the ontological point of view.

In this method we omit relations between the terms and the resulting CMoD is a flat controlled vocabulary and may be represented just using the set of terms.

5.2 Complete Ontological Analysis

In this method we create a (more or less) complete CMoD including the relations. Conceptual maps then serve not just as an identity maintaining device, but as a stand-alone knowledge formalisation and representation technique that helps to understand, record, share, communicate and validate knowledge of the domain. Models of the domain are then built upon the previously created CMoD(s).

Figure 5 depicts the position of ontological analysis and the created CMoDs in the context of Ambler's software development process ([1]). Ontological analysis is performed during the *Initate* and *Construct* phases, whereas the resulting concept maps may be utilised during the whole project life cycle[7]. In the typical iterative development process, OA will be performed during every iteration.

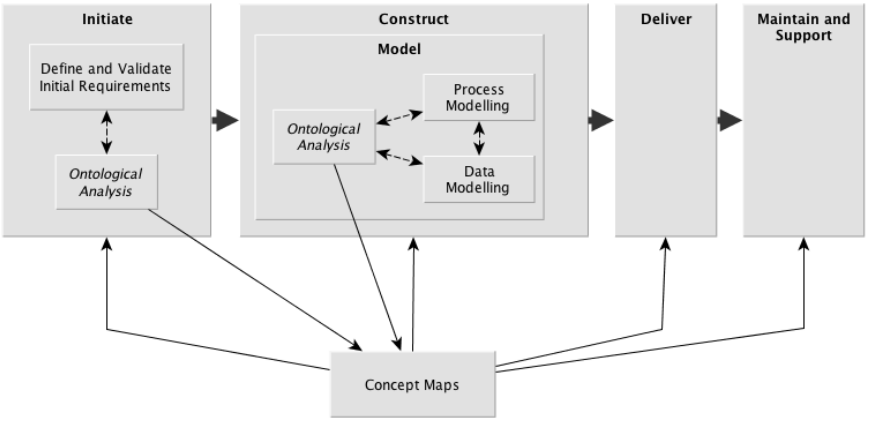


Fig. 5. Ontological analysis in the software development process

A simple example of CMoD is in Figure 6. It represents a (partial) ontology of an internal company supply store. We denote the terms by unique capital letters enclosed in square brackets and relations between the terms by unique

[7] Concept maps may be updated during any stage of course, e.g. if an error or ambiguity is detected.

small letters enclosed in round brackets, so that we may easily express the *representationOf* relations.

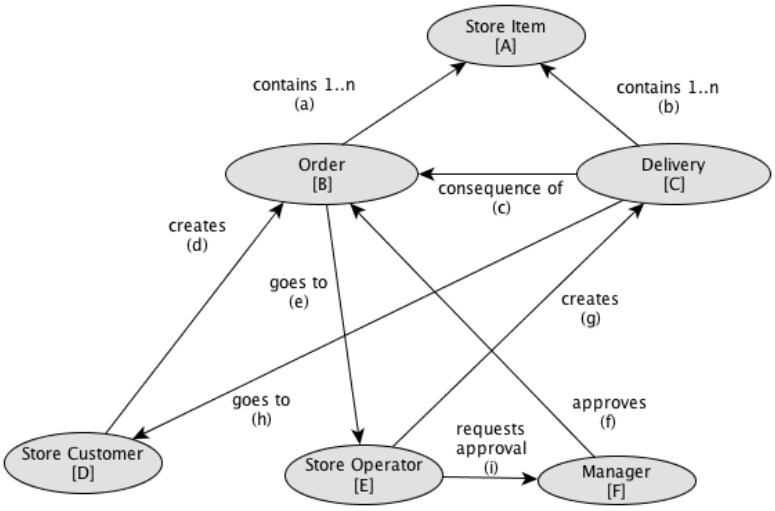


Fig. 6. Example: ontological analysis of an internal company supply store

This CMoD may be have been created during the requirements gathering and initial modelling sessions with the future IS customer. Standard BORM analysis and modelling then follows (refer to [**?**] for details). During this analysis, the Business Architecture diagram (Figure 7) is created that depicts the process architecture.

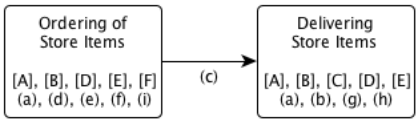


Fig. 7. Business Architecture diagram of the example

There are two scenarios identified and a transition relation between them. The *representationOf* relations with the elements of the CMoD are presented. As we may see, all the CMoD elements and relations are covered, i.e. this is a surjective mapping. This is not a coincidence. The mapping represented by the *representationOf* relations **should be always surjective**. We may informally "prove" this theorem using absurdum proof: If the mapping is not surjective, it would mean that there exists at least one element t in the CMoD such that

there is no mapping $represenatationOf(e, t)$, where e is some element of the BA diagram[8]. This would mean that the domain term represented by the element t is not related to any scenario and thus would be needless. This represents a simple **inter-layer consistency check**. If the mapping happens to be non-surjective, then either:

- The term is needless and should be thus removed from the CMoD.
- Or the term *is* crucial and thus represents some requirement that should result in relating it to some scenario.

Next we may elaborate the ORD diagram for each scenario (Figure 8 and Figure 9).

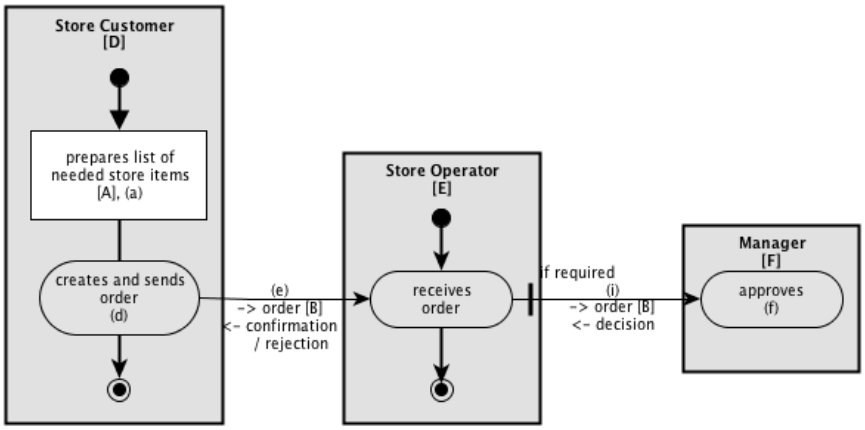


Fig. 8. ORD of the scenario "Ordering of Store Items"

Again, the elements and relations of the diagrams are related to the elements and relations of the CMoD. Moreover, the set of inter-layer relations for each diagram should be equal to the set of relations of the corresponding scenario. This provides a kind of **elements transistion check** (see [16] for a discussion about the importance of maintaining transitions between model concepts)[9].

Figure 10 finally shows the class model (without attributes) of the example and its *representationOf* relations.

As we may see, every diagram adds some details that are not depicted in the CMoD – ORD diagrams add process and collaboration details, while class diagrams add details about the relations and attributes (which we ommited in Figure 10) – and at the same time it omits some terms and relations from the CMoD as it represents a certain limited view.

[8] Or the union of all BA diagrams if there are more in the model.
[9] The absurdum "proof" may be done here as well.

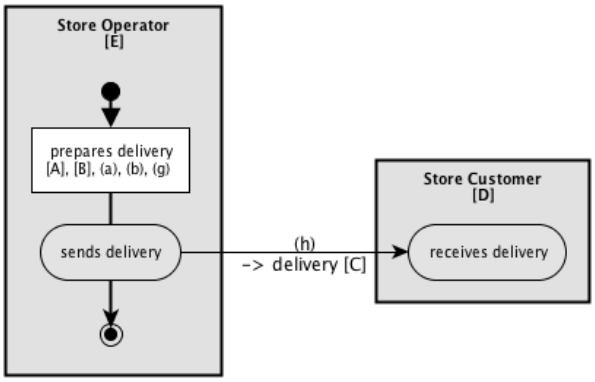


Fig. 9. ORD of the scenario "Delivering Store Items"

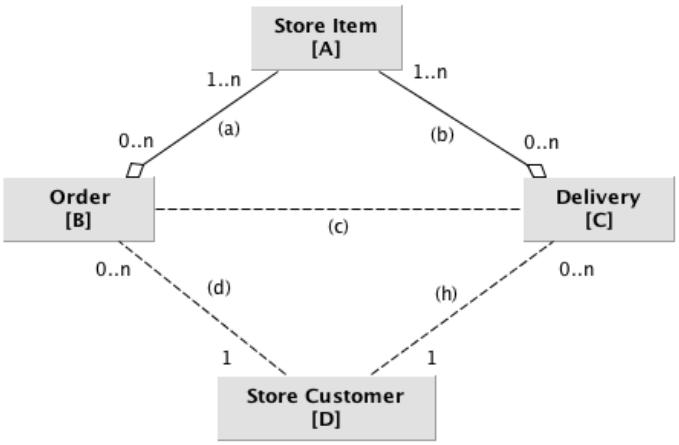


Fig. 10. Class diagram of the example

6 Discussion and Conclusions

Ontological analysis may be performed in an "agile style" by just building the *controlled vocabulary* (subsection 5.1) along with the usual modelling. This approach means minimum overhead and brings the support for

– identity maintaining,
– models refactoring,
– consistency checks,
– elements transistions checks,
– impact analysis during the change management.

If we perform a complete ontological analysis (subsection 5.2), i.e. we analyse also relations between the term, we deepen the above benefits. Apart from them, a complete ontological analysis may take advantage of wide variety of ontology engineering tools for the created concept maps: validations, reasoning, model transformations, knowledge bases utilisation, prototyping, problem-solving and inference tools and others. On the other hand, a thorough ontological analysis increases elaborateness of analysis phase and brings higher demands on analysts.

As for the model refactorings support and consistency maintenance, we would propose the following two approaches that may be implemented into a CASE tool:

- If e is a model element, t is the CMoD element (domain term) and there holds $representationOf(e, t)$, then the name of e would be taken from the name of t (let us denote $e.name = t.name$). This may not, however, be always suitable, thus assigning a flag to the *representationOf* relation that would be used to switch this feature on and off as needed would be necessary.
- If the name is not taken, whenever $t.name$ is changed, the user is presented with the list of the related elements (i.e. the set of elements $E = e_1, e_2, \ldots, e_n$, where there exists $representationOf(e_i, t)$) that may be affected by this change. The same would happen if the user is about to remove the term t from the CMoD.

The BORM method distinguishes between the terms *Participant* and *Participant Role*, where Participant is an entity from the domain that plays (typically several) roles (like "performs", "is informed", "is responsible", etc.) in various scenarios and it may even play several roles in one scenario. This distinction is very useful, because it enables us to keep track of the Participant identity related to its several roles. However, if we introduce the concept of maintaing the *representationOf* relations between the layers M0 and M1 (Figure 3), we may omit the concept of *Participant*.

Ontological analysis concept supporting the modelling process was presented on the BORM methodology, however it may be used for any modelling method and notation, like UML, OMT, SA/SD and others and similar benefits may be expected.

7 Related Work

Craft.CASE is the most advanced CASE tool for the BORM method[10] known to the author. Some glimpses of the concepts presented in this paper are implemented in the tool:

- The Sketch tool supports drawing of free elements, which may be used to draw concept maps, however, their utilisation in the future modelling steps is limited.

[10] Precisely speaking, Craft.CASE implements the C.C method, which has however the same syntax of the diagrams and the semantics differs just in slight details.

– Craft.CASE maintains the identity of data flows by putting them into a controlled vocabulary, which ensures consistency when renaming and this concept is also succesfully utilised in reporting.
– Craft.CASE elements follow the concept of Model-View. One element may be copy-pasted into several places while maintaining a simple identity. This assures automated renaming and property changes, however extensive use is limited by the absence of the controlled vocabulary.

These features bring considerable advantages over the tools that do not support them, but unfortunatelly, they are just separate ideas without the proper formal and methodological framework, which limits their benefits and also brings some unpleasant side-effects[11].

8 Future Work

The research continues both on the formal, methodological and practical levels. One vast topic is the method of creating the concept maps and related questions: How to identify the crucial terms? Which relations should be captured and which should be omitted? Is it possible to make some standardized set of relations? ... Many answers to the ontological analysis questions and methods may be found in the existing literature about ontological engineering, however they are usually too general or directed towards knowledge management or semantic web development, not to enterprise systems modelling.

Another field is utilisation of the presented concepts and methods. Some of the most imporant utilisation possibilities were listed above, however there are definitely many more: e.g. we are now working on the multi-language support for modelling using concept maps.

And last but definitely not least – practical utilisation of the presented concepts requires a quality CASE tools support. This is probably the most appealing topic and we are working on a CASE tool prototype for the BORM method that would incorporate ontological analysis as described, and bring its benefits into practice.

Acknowledgements

This contribution was elaborated with a support of grant no. 201011130043 of Grant Agency of The Faculty of Economics and Management of the Czech University of Life Sciences in Prague.

[11] Mostly with the respect to the C.C scripting language implemented in the tool for elements manipulations.

References

1. Ambler, S.W.: *Process Patterns : Building Large-Scale Systems Using Object Technology*, Cambridge University Press (1998).
2. Ambler, S.W.: *The Object Primer – Agile Model-driven Development with UML 2.0*, Cambridge University Press (2005).
3. Atkinson, C., Kühne, T.: "Model-driven development: A metamodeling foundation", *In: IEEE Software*, Vol. 20, no. 5, pp. 36-41 (2003).
4. Baldwin, D.: "Software Development Life Cycles: Outline for Developing a Traceability Matrix", The Regulatory forum, available online at `http://www.regulatory.com/forum/article/tracedoc.html`
5. Barr, A., Feigenbaum, E.A. (eds.): *The Handbook of Artificial Intelligence*, William Kaufmann, Los Altos, CA (2003).
6. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: "What are ontologies, and why do we need them?" *In: IEEE Intelligent Systems*, Vol. 14, no. 1, pp. 20-26 (1999).
7. Fowler M.: *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)*, Addison-Wesley Professional, ISBN 978-0321193681 (2003).
8. Frankel, S.D.: *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley, New York (2003).
9. Gasevic, D., Djuric, D., Devedzic, V.: *Model Driven Engineering and Ontology Development (2nd Edition)*, Springer (2009).
10. Guizzardi, G. "The Role of Foundational Ontology for Conceptual Modeling and Domain Ontology Representation", Companion Paper for the Invited Keynote Speech, 7th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania (2006)
11. Gruber, T.R.: "A translation approach to portable ontology specifications", *In: Knowledge Acquisition*, Vol. 5, no. 2, pp. 199-220 (1993).
12. Hendler J.: "Agents and the semantic web", *In: IEEE Intelligent Systems, Vol 16, no. 2, pp. 30-37* (2001).
13. Knott, R.P., Merunka, V., Polk, J.: "The BORM methodology: a third-generation fully object-oriented methodology", *In: Knowledge-Based Systems*, Vol. 16, no. 2, pp. 77-89 (2003).
14. Knott, R.P., Merunka, V., Polak, J.: "The BORM Method: A Third Generation Object-Oriented Methodology. In Management of the Object-Oriented Development Process", ed. Liping Liu, and Borislav Roussev, ISBN 9781591406044, 337-360 (2006).
15. Molhanec, M. "STEP standard - Perspective and futurity". In 29TH INTERNATIONAL SPRING SEMINAR ON ELECTRONICS TECHNOLOGY, 495-499. International Spring Seminar on Electronics Technology, ISSE. 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE. ISBN 978-1-4244-0550-3 (2006).
16. Picka, M., Pergl, R.: Gradual modeling of information system: Model of method expressed as transitions between concepts. *In: ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration*, pp. 538-541, Paphos, Cyprus (2006).
17. Selic, B.:"The pragmatics of model-driven development", *In: IEEE Software*, Vol 20, no. 5, pp. 19-25 (2003).
18. Sowa, J.F.: *Knowledge Representation: Logical, Philosophical and Computational Foundations*, Brooks Cole, Pacific Grove, CA (2000).

19. Swartout, W.R., Tate, A.: "Guest editors' intruduction: Ontologies", *In: IEEE Intelligent Systems*, Vol 14, no. 1, pp. 18-19 (1999).
20. Wiegers K. E.: "Software Requirements", Second Edition, Microsoft Press (2003).

Chapter 11

Instance-Level Modelling and Simulation Revisited

[194] Pergl, R.; Sales, T. P.; Rybola, Z. Instance-Level Modelling and Simulation Revisited. In Enterprise and Organizational Modeling and Simulation, Valencia, 281 Spain: Springer, June 2013, ISBN 978-3-642-41637-8, pp. 85 – 100.

Instance-Level Modelling and Simulation Revisited

Robert Pergl, Tiago Prince Sales, Zdeněk Rybola

Department of Software Engineering,
Faculty of Information Technologies,
Czech Technical University in Prague, Czech Republic
{robert.pergl,zdenek.rybola}@fit.cvut.cz

Ontology and Conceptual Modeling Research Group (NEMO),
Computer Science Department,
Federal University of Esprito Santo, Brazil
tpsales@inf.ufes.br

Abstract. Instance-level modelling is a sort of conceptual modelling that deals with concrete objects instead of general classes and types. Instance-level modelling approach offers a rather innovative way for communication with domain experts extremely useful for them, as they can see their real data in the context of the given model. Various approaches were presented in the paper "Instance-Level modelling and Simulation Using Lambda-Calculus and Object-Oriented Environments" at EOMAS 2011. The present paper is a sequel and it presents additional approaches we find useful in practice: Fact-oriented modelling, OntoUML in combination with OCL and the Alloy and Eclipse-based framework Dresden-OCL. We present key features of the various approaches and demonstrate them on a running example, we follow up with a discussion comparing these approaches. Notice that OntoUML combined with the Alloy is an original research achievement built on the research of OntoUML.

Key words: Instance-level modelling, OntoUML, Alloy, Fact-Oriented Modelling, ORM, DresdenOCL

1 Introduction

Conceptual modelling as defined in [1] is an activity of describing some aspects of a domain of interest formally for understanding and communicating. The activity leads to a conceptual model, which should be used and understood by humans.

The quality of the results (i.e., the conceptual model) can be evaluated with respect to various aspects, some of which concern the used language, other concerning the models itself [2], [3]. To obtain high-quality conceptual models, several factors are in play:

– How is the model powerful with respect to the given domain and the goal of the modelling.

– Correct understanding of the domain structure, facts and rules, as perceived by the modeller.
– Understanding of the model by the domain expert.

The last point is rather crucial as it is a necessary condition for model *validation*. We understand validation as an activity performed by modellers and domain experts to evaluate if the model accurately captures the expert's view of the domain concepts and if all the necessary concepts and properties were modelled.

We all know that business users have issues dealing with technical models [3]. Given that, their model validation is not optimal: if the domain expert does not fully understand the contents and especially the relations in the model, the validation they provide covers the model only partially.

Software engineers' experience shows that users are able to validate the model fully only once they put their hands on the software product [4]. This causes rather serious problems for software engineers: an error identified in the resulting product is way more expensive to fix than errors discovered in earlier stages of the software development [5].

This makes instance-level modelling a very promising approach for software engineering, knowledge engineering and generally any efforts of domain conceptualisation, because its main concern is to identify domain constraints, in particular those that are not easily captured by conceptual modelling notations. This is fully explained in [6]: Merunka discusses lambda calculus utilisation for instance-level modelling, i.e. computation and manipulation aspects of instances. In this paper, we would like to follow his lead and discuss instance-level modelling from the perspective of *structural modelling*, which – in fact – logically precedes Merunka's topic. Structural modelling provides a solid back-bone for any further model development and operations upon the model.

2 Goals and the Structure of the Paper

The goal of the paper is to provide modellers with various approaches to structural instance-level modelling.

The structure of of the paper is as follows: Section 3 provides an overview on instance-level modelling. Section 4 presents three selected approaches for comparison: the fact-oriented modelling approach represented by ORM; OntoUML with combination of OCL and Alloy (our original contribution); and the Dresden OCL approach. Section 5 presents running examples implemented using all three approaches. After that, we discuss and compare features of each method in Section 6. Finally,Section 7 presents our conclusions and future works.

3 Instance-level Modelling and the Quality of Model

Let us just briefly summarize the importance of instance-level modelling in the conceptualization process. We will provide a complete thought flow; the essential statements are in bold below.

As Birkmeier and Overhage comment in [3], "the quality of conceptual models is influenced by a variety of factors". Hadar and Soffer present a scheme depicted in Figure 1 showing modelling process as one of the key factors.

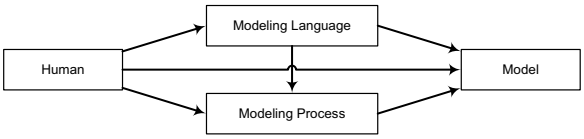


Fig. 1. Factors that affect the quality of a conceptual model [7]

The modelling process in the context of Ullman's triangle [8] (Figure 2) means to perform *abstraction* to create *conceptualization* from the *reality*. To ensure that the conceptualization is correct, we need to *validate it* with the domain expert [9]. If the domain expert understands the *language* in which the conceptual model is written (that is, s/he understands both the *syntax* and the *semantics*), we may expect that the validation is performed dutifully. However, business users usually have issues with technical notations and abstractions, as may be seen e.g. in [3]; thorough validation is hence close to impossible.

As Merunka suggests in [6], a perfect solution for non-technical users to talk about their domain is instance-level modelling. This may be achieved using the three following ways:

1. Asking the domain expert to present some samples of instance-level data – this approach is used for conceptualization e.g. in [9].
2. Generating instance examples based on the conceptualization and asking the domain expert to confirm that the conceptualization expresses the domain – this approach is discussed in [6] and may be also achieved by Alloy, as discussed in subsection 4.3.
3. Generating instance counter-examples to eliminate conceptualization leaks, i.e. that it is not possible to generate instances that are not aligned with the domain; the Alloy approach can achieve that, see subsection 4.3.

4 Selected Approaches

First, there are several approaches discussed in [6]: BlueJ, .NET Object Test Bench and, finally, author's original lambda-calculus based approach. The first

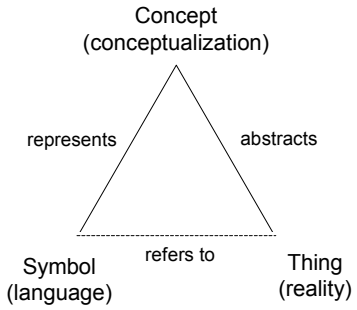


Fig. 2. Ullman's triangle [10])

two mentioned (BlueJ and .NET Object Test Bench) are focused on UML and programmers' needs and, hence, they do not deal with the ontological level of the model. Merunka's approach, on the other hand, targets calculations and querying. In the rest of this section, we would like to discuss some other instance-level modelling approaches that are focused on structural aspects. For this paper, we selected four of them: traditional Fact-Oriented Modelling, OntoUML, Alloy and OCL. They are all based on instance-level modelling, however, they differ in their characteristics and their original purpose. In the running example (Section 5) we present our own original approach that connects OntoUML + OCL + Alloy.

4.1 Fact-Oriented Modelling

Fact-oriented modelling is one of the traditional approaches within instance-level modelling. It was formulated by Terry Halpin in his Ph.D. thesis in 1989, its roots, however, reach to 1970s. Fact-oriented modelling - as described by Halpin in [9] - is a conceptual approach to information modelling and information systems engineering. It was designed to promote correctness, clarity, and adaptability. This approach enables us to model, transform, and query information in terms of the underlying *facts* of interest, where facts and rules may be verbalized in language that is readily understandable by non-technically-minded users from the business domain. In contrast to UML-based modelling (e.g. OntoUML, see subsection 4.2), fact-oriented models are attribute-free: they treat all facts as relationships (unary, binary, ternary etc.). For example, the following fact types `Person smokes`; `Person was born in Country` are used to replace the attributes `Person.isSmoker` and `Person.birthCountry`.

Still, the most popular fact-oriented approach is probably **Object-Role modelling (ORM)**. It got its name because it pictures the world in terms of objects (entities or values) that play *roles* (parts in relationships). There is a number of closely related dialects, all of which use a similar object-role graphical notation and there are also approaches adopting different graphical notations (for more details refer to e.g. [9]).

4.2 OntoUML

OntoUML is an ontologically well-founded conceptual modelling language that aims to describe structural aspects of a given domain of interest [10]. An OntoUML model is understood as a reference conceptual model, which is used mainly to achieve semantic interoperability between agents, both human and artificial. As it is a very promising modelling language used in practice successfully, we will discuss its fundamentals and its relation to instance-level modelling.

The main concern of OntoUML language is to capture a conceptualization of a community regarding the given domain of interest in the most precise way. To achieve that, the language was designed to reflect the Unified Foundational Ontology (UFO) concepts [10]. UFO, then, is a domain independent system of categories, which addresses ontological structural aspects of individuals and universals, such as instantiation, rigidity, identity and dependence.

The language was designed as an UML Profile to be more accessible to users. Each stereotype is embedded with constraints; this, then, provides precise meaning for the concepts and it restricts the way in which they can be combined to develop a model.

OntoUML has been successfully employed in a number of projects in several different domains, from Petroleum and Gas [11] to News Information Management [12] and data center IT architecture [13]. In fact, after a significant number of successful applications in real-world engineering settings [14], it has been recently considered as a possible proposal to the OMG SIMF (Semantic Information Model Federation) standardization request call [15].

4.3 Alloy

Alloy [16] is a language developed for describing structural properties. It was created by Daniel Jackson at MIT. It is a declarative, first-order logic language based on set theory. The language is supported by an instance level-oriented solver named Alloy Analyzer: the analyzer takes the constraints of a model and tries to find structures that satisfy them.

An Alloy specification consists of logical constraints defined in signature and fact declarations. When a specification is instantiated by the Alloy Analyzer, atoms are generated from signatures. Importantly, the atoms respect the logical constraints in the model. In other words, a signature at the model level introduces a set of atoms at the instance level.

The analysis made by the Alloy Analyzer is based on the SAT (boolean satisfiability) technology. The Analyzer translates constraints from Alloy into Boolean formulas, which are then fed to an off-the-shelf SAT solver. The analysis can be performed in two different ways: first, as a model exploration, through example generation; and second, as a property checking, through search of counter examples.

Alloy has been applied as a validation tool for different purposes such as an analysis of UML models [17], verification of Java Code [18], and for verification of i* models [19].

To help modellers in defining the necessary instance-level constraints and to understand their modelling decisions, the OntoUML language is supported by a validation tool, named OntoUML2Alloy [20]. This tool automatically translates OntoUML models into an Alloy specification and it allows the modellers to implement their OCL constraints directly in Alloy.

4.4 OCL and DresdenOCL

Object Constraint Language – OCL [21] – is a specification language used to delimit restrictions for a UML model that cannot be expressed directly in the UML notation. OCL is a part of the UML standard [22]. It is used to define invariants (conditions that must be satisfied by all instances of the element), pre- and post-conditions of element methods and it can be even used as a query language.

DresdenOCL [23] is a toolkit for modelling OCL constraints and their transformations and interpretations. DresdenOCL is distributed as a stand-alone library as well as a plugin to Eclipse IDE. The toolkit is capable of working on various models, like UML, EMF and Java. It provides syntax checker for OCL constraints and it also interprets the constraints in the context of a loaded model and a model instance. Therefore, the toolkit can be used for simulation and model validation; as a model instance it can be created with real object samples and evaluated against the model and constraints. The toolkit also provides tools for model transformation and source code generation along with the OCL constraints into SQL and Java/AspectJ.

5 A Running Example – Book Publishing

In this section, we present a running example from a domain of book publishing. We want to demonstrate the conceptualization possibilities of the approaches we discussed above and, in particular, we want to focus on a quality of the given model and its instance-level modelling features. The example originally comes from [9] where it is modelled in ORM. We like the example, especially because it is not trivial, it contains several "tricky" facts that can appear in practice and that - we believe - cannot be expressed easily without instance-level modelling.

5.1 ORM

Figure 3 shows am ORM schema for the given domain. As already mentioned, the example (alongside with the following description) was adopted from [9]. We refer the reader to a detailed description of how to develop such a schema cooperating with the domain expert to the original paper.

Each `book` is identified by an International Standard Book Number (`ISBN`), each `person` is identified by a `person number`, each `grade` is identified by a `grade number` in the range 1 through 5, each `gender` is identified by a `code`

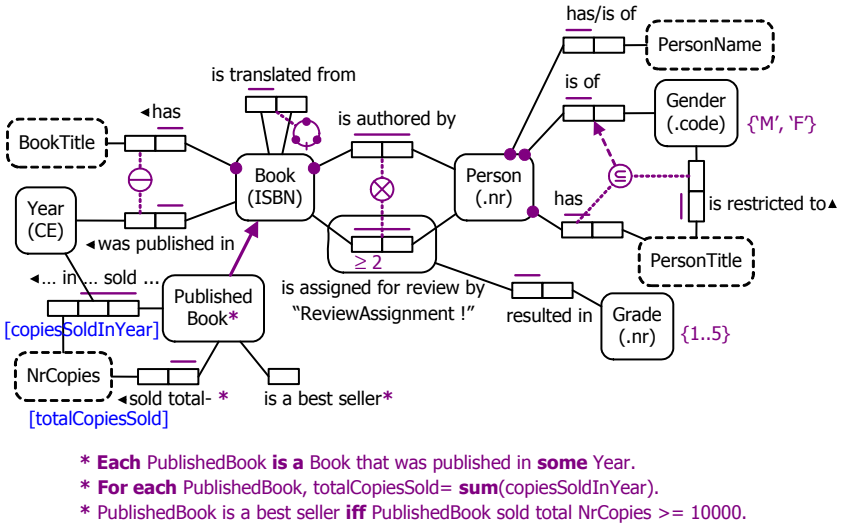


Fig. 3. ORM version of the book publishing conceptual model

(`M` for male and `F` for female), and each `year` is identified by its common era (`CE`) number. `PublishedBook` is a derived subtype determined by the subtype definition shown at the bottom of the figure. `ReviewAssignment` objectifies the relationship `Book is assigned for review by Person`, and is independent, since an instance of it may exist without playing any other role (one can know about a review assignment before knowing what grade would result from that assignment).

The internal uniqueness constraints (depicted as bars) and mandatory role constraints (solid dots) are verbalized as follows: Each Book is translated from at most one Book; Each Book has exactly one BookTitle; Each Book was published in at most one Year; for each Published Book and Year, that PublishedBook in that Year sold at most one NrCopies; Each PublishedBook sold at most one total NrCopies; It is possible that the same Book is authored by more than one Person and that more than one Book is authored by the same Person; Each Book is authored by some Person; It is possible that the same Book is assigned for review by more than one Person and that more than one Book is assigned for review by the same Person; Each ReviewAssignment resulted in at most one Grade; Each Person has exactly one PersonName; Each Person has at most one Gender; Each Person has at most one PersonTitle; Each PersonTitle is restricted to at most one Gender.

The external uniqueness constraint (circled bar) indicates that the combination of `BookTitle` and `Year` applies to at most one `Book`. The acyclic ring constraint (circle with three dots and a bar) on the book translation predicate indicates that no book can be a translation of itself or any of its ancestor translation sources. The exclusion constraint (circled cross) indicates that no

`book` can be assigned for `review` by one of its `authors`. The frequency constraint (≥ 2) indicates that each `book` that is assigned for review is assigned for `review` by at least two `persons`. The subset constraint (circled subset symbol) means that if a `person` has a `title` that is restricted to some `gender`, then the `person` must be of that `gender`. The first argument of this subset constraint is a `person-gender` role pair projected from a joint path that performs a conceptual join on `PersonTitle`. The last two lines at the bottom of the schema declare two derivation rules, one specified in attribute-style using role names and the other in relational style using predicate readings.

5.2 OntoUML+OCL+Alloy

Figure 4 depicts the book publishing domain example in OntoUML.

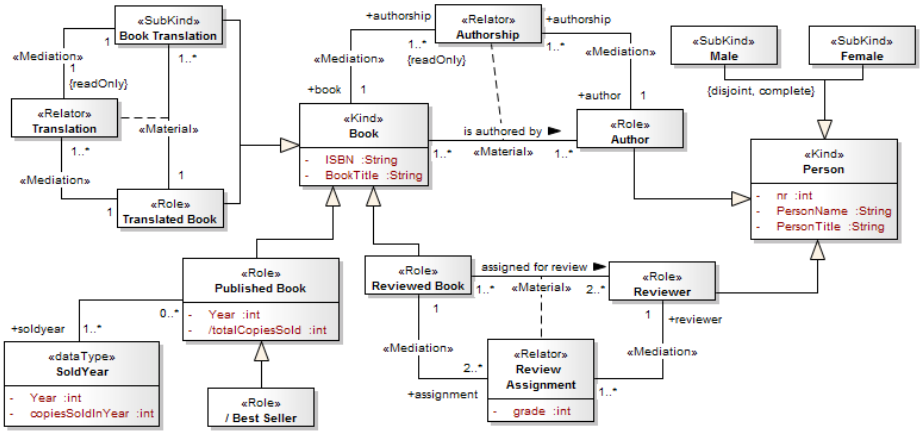


Fig. 4. OntoUML version of the book publishing conceptual model

Expressing type-level constraints in OntoUML The *kind* stereotype represents rigid types, i.e., their instances must always instantiate them in every possible scenario. It also provides identity principle for its instances and thus, no objects can instantiate two *kinds* simultaneously. Two *kinds* are identified in this model: `Person` and `Book`. *Kinds* may be specialized by *subkinds*; these are also rigid types, but that don't provide identity. *Subkinds* always contain individuals that share the same identity principle and thus, *subkinds* may only specialize one *kind*. Hence, `Man`, `Woman` and `Book translation` are *subkinds* in the model.

Roles represent anti-rigid types which are relationally dependent. Anti-rigidity means that for every individual who is an instance of a *role* in a given moment, there is at least another moment (either in the future or in the past)

in which that individual is not an instance of the *role* anymore. Relation dependency means that for an object to be an instance of a *role*, it must be related to at least one other object. In the example, this concept enables modellers to express that `Person` can become `Authors` and `Reviewers` when they have an authorship on a `Book` or when they are assigned to `review` a book, respectively. And that a book can be translated if there is another `book` that is its translation.

The relations that characterize *role* are objectified by *relators* in OntoUML. This – also rigid – type is externally dependent on the composing *roles*; that means that the individuals that are related through an instance of a *relator* may never change during the existence of the *relator*. For example, for every instance of `Review Assignment`, the `Book` and the `Person` are always the same. *Relators* have a complementary restriction: every instance of a *relator* must mediate at least two distinct individuals, for example, the `Translation` forbids the existence of `Books` which are a `translation` of themselves at the same time.

Even though the running example does not contain this construct, we would like to mention the *phase* type, as it is closely related to the instance-level modelling. The *phase* constraint states that for every *phase* partition, an instance might instantiate only one *phase*. However, a *kind* – like a `Person` - can have different *phase* partitions, like:

- A phase partition regarding people's age; it contains the *phases:* `Child`, `Adult` and `Elder`.
- Another *phase* partition regarding their health containing the *phases:* `Sick Person` and `Healthy Person`.

In this example, hence, it is possible for an instance of a `Person` to be both a `Child` and `Sick`, or an `Adult` and `Healthy`. But an instance can never be `Sick` and `Healthy` at the same time, as much as it cannot be a `Child` and an `Adult` simultaneously.

Other examples of such instance-level constraints would be weak supplementation for meronymic relations: a whole must have at least two disjoint parts; the *relator* rule described in the previous section which states that a *relator* must mediate at least two distinct individuals.

Expressing Instance-Level Constraints in OCL Although the graphical notation allows modellers to express many important ontological distinctions, it is not sufficient for expressing all instance-level restrictions, such as `a Person may not review their own books`. To achieve that, the models must be enriched with OCL rules. For this restriction, a possible OCL invariant would be:

```
context _'Reviewed Book'
inv noAuthorReviewHisBook :
  self.oclAsType(Book).authorship.author->asSet()->
  excludesAll(self.assignment.reviewer->asSet())
```

Another enforced domain restriction is that every instance of `Book` which plays the role of `Published Book` is not related to two or more instances of `SoldYear` which have the same `Year` attribute.

```
context _'Published Book'
inv : self.soldyear->isUnique(Year)
```

The objectification of some relations through the relators improves expressivity of the model, since it allows for a representation of additional cardinalities. For example, it allows expressing that a `Review Assignment` assigns a single `Book` to exactly one `Reviewer` - and not many, as it could have been interpreted. On the other hand, it may be necessary to express that certain elements should not be related twice by different *relators*. So, for the `Authorship` *relator*, it is necessary to state that every `Person` is an `Author` of a `Book` only once: this requires to express that an `Author` is related to exactly one `Authorship` *relator* for every `Book` which he is an author of.

```
context _'Book'
inv : self.authorship->isUnique(author)
context _'Author'
inv : self.authorship->isUnique(book)
```

For derivation attributes, such as `totalCopiesSold` of `Published book`, it can specified as:

```
context _'Published Book'::totalCopiesSold:int
derive : self.soldyear.copiesSoldInYear->sum()
```

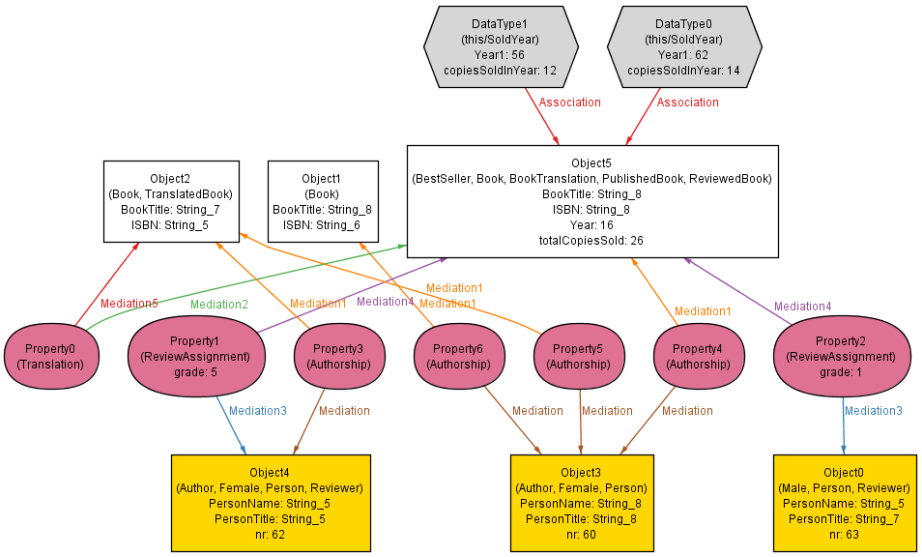


Fig. 5. Possible instantiation of the book publishing OntoUML model in Alloy

Identifying instance-level constraints in Alloy OntoUML modellers may use the Alloy tools to visualize the consequences of their modelling decisions. An

example generated in Alloy represents a set of possible Worlds according to the conceptual model. Figure 5 is a possible instantiation of the running example in Alloy.

Notice that we did not set the `Best Sellers` as those books that were sold more than 1000 times by chance. Since there was no restriction associated with it, the validation tool generated an example in which a book was a best seller after having sold only 26 copies. This example shows how the supporting tools help users to identify missing constraints in their models – even though OntoUML does not provide instance-level restriction constructs.

5.3 DresdenOCL implementation

DresdenOCL toolkit [23] can be used to create a conceptual model in pure UML enriched with a set of OCL rules. In addition, the tool allows users to populate their model by manually created instances and it allows the users to check whether this set of instances complies with the model specification. On top of that, the tool can be used to generate SQL or Java/AspectJ source codes of the application, and this, too, can be used to specify model instances.

UML model definition with OCL constraints The UML version of the running example, developed in the MDT UML2 plugin in Eclipse, is shown in Figure 6. Note that there are some simplifications compared to the OntoUML version, since UML does not support imposing all the constraints on the model. The presented model can be easily loaded in DresdenOCL Toolkit in Eclipse.

In the running example, a rather tangible difference between the OntoUML and the UML version is that, in the latter, there is no notion of modality. All types in UML are rigid; it means that if an object is an instance of a `Book`, it cannot be changed to be an instance of a `Publishment`. This is in a stark contrast to the *Role* classes in the OntoUML model: here, if an object needs to change its type, it must be destroyed and created as an instance of another type.

Another clear difference is that, in UML, material relations do not require the representation of relators. They are employed, if the modeller wants to have at least one attribute. This is the case for the `Review` class: it has the attribute `grade` – and it is not the case for the `Authorship` (between `Person` and `Book`) and `Book Translation` (between `Book` and `Translated Book`).

Finally, the `Publishment` has been modelled by a composition of classes `Book` and `Publishment`, not by a *role* specialization.

To mantain the restrictions given by the domain, OCL constraints must be defined as discussed in Section 5.2. In DresdenOCL, the constraints are stored in a separate file loaded to the tool along with the model. In addition to the constraints discussed above, there are additional constraints - that must be defined - to express the restrictions defined by the OntoUML constructs themselves. For instance, in the UML model, we do not restrict the amount of reviews per book to two because we do not have the role `Reviewed Book`. Therefore, an additional constraint must be defined as follows:

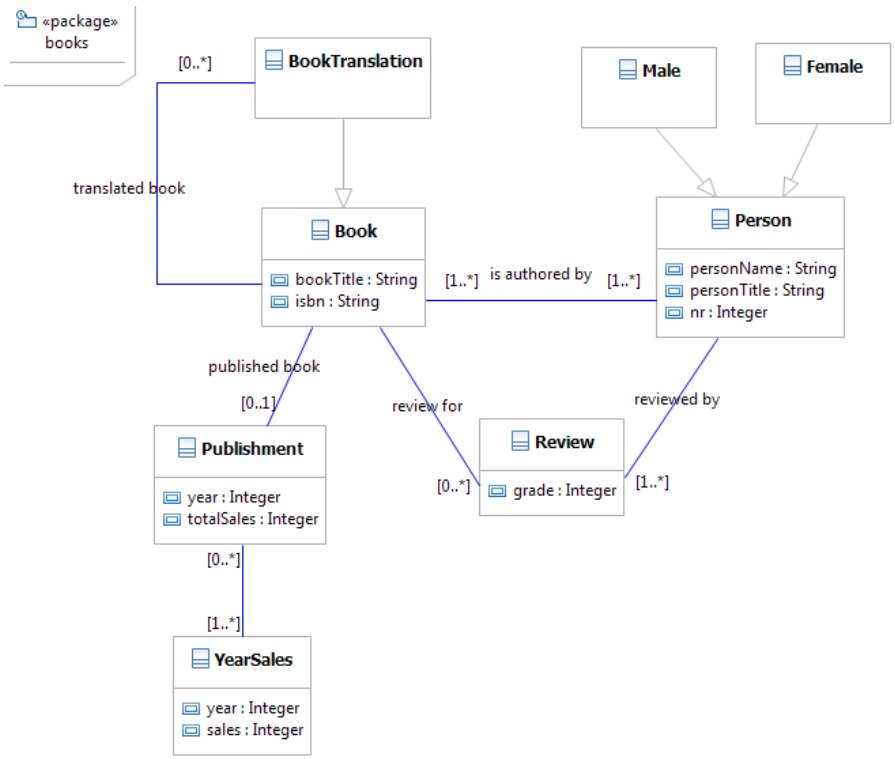


Fig. 6. UML version of the book publishing conceptual model

```
-- no or at least two reviews for each book
context Book inv noOrTwoReviews:
if self.reviews->size() > 0
  then self.reviews->size() >= 2
  else true
endif
```

Model simulation and validation DresdenOCL toolkit can be used for model validation and simulation: an instance of the model can be evaluated against the OCL constraints. The model instance can be loaded from an EMF Ecore-based model instances, Java class files or XML files. An example of a Java class used for the definition of the running example's instance is shown in Figure 7. The model instance defines the same objects identified in Figure 5 with the `object4` being a reviewer for one of his own books.

When the model and the model instance is loaded to the toolkit, OCL constraints can be interpreted to validate the model instance. Figure 8 shows the Eclipse perspective with the constraints, loaded model instance and OCL interpretation results. In the lower left-hand side panel, there are the results for

```
public class BookInstanceProvider {
  public static List<Object> getModelObjects() {
     List<Object> result = new ArrayList<Object>();
     Person person0 = new Male();
     person0.setPersonName("String_5");
     person0.setPersonTitle("String_7");
     person0.setNr(63);
     result.add(person0);
     ...
     Book book1 = new Book();
     book1.setBookTitle("String_8");
     book1.setIsbn("String_6");
     result.add(book1);
     ...
     person3.getBooks().add(book5);
     book5.getAuthors().add(person3);
     ...
     return result;
  }
}
```

Fig. 7. An example of a model instance definition for DresdenOCL toolkit in a Java class

each OCL constraint interpreted in all possible contexts. A violation is detected by the *false* result for one of the reviews: the reviewer is also one of the book authors.

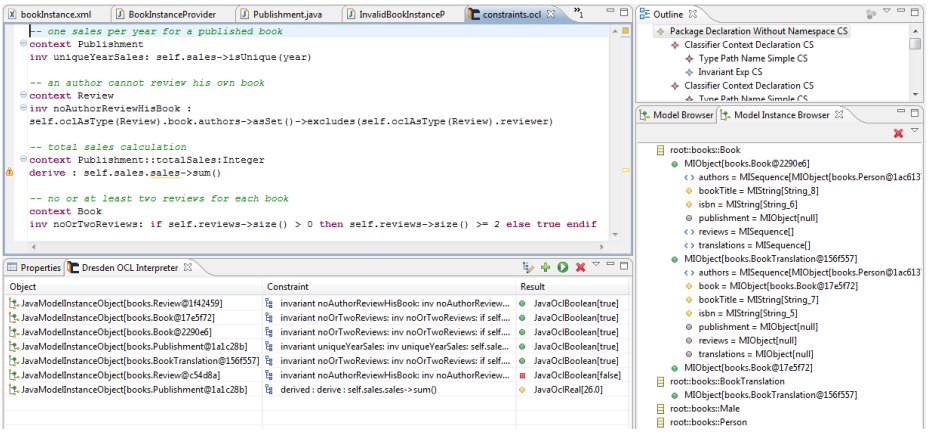


Fig. 8. The Eclipse perspective with the OCL constraint, the loaded model instance and the OCL constraints interpretation results

6 Discussion and Related Work

ORM represents a very precise and highly expressive approach. A lot of complex instance-level constraints may be formalized – the diagram in Figure 3 covers almost all instance-level constraints. On the other hand, ORM is based exclusively on mathematical constructs (set theory, predicate logic, relations and their characteristics) - and this is its main disadvantage. As Guizzardi explains in [10], the mathematics (formal semantics) does not guarantee ontological and cognitive consistency. In fact, set theory may create ontological extravagances not aligned with our human real-world cognition.

Avoiding attributes in the fact-oriented modelling enhances *semantic stability*, as noted in [9]. For example, if we used a `birthCountry` attribute and then later decided to record the population of countries, we would need to remodel the information as a relationship and recode all the queries based on it. As user requirements may change during the project [4], this may turn out to be a great benefit. An attribute-free approach is highly instance-oriented: it enables all fact structures to be easily populated with fact instances. For more details on fact-oriented modelling and its comparison to attribute-based approaches, see e.g. [9].

OntoUML is focused on producing ontologically well-founded models – it combines mathematical constructs with cognitive science. It certainly encompasses several instance-level constructs (e.g. an individual may instantiate only one *phase* in a given *phase* partition, which may change over time) and instance-level models may be generated from it using the simulation in Alloy. However, OntoUML is not entirely focused on instance-level modelling and it lacks some of the required constructs. These, of course, may be added using additional mechanisms, like the OCL rules presented in the running example.

To further help modellers to discover the missing instance-level constraints, a set of semantic antipatterns have been identified for OntoUML in [24]. Semantic antipatterns are recurrent modelling decisions that - even though logically valid - are prone to produce mismatches between the represented and the intended instances of the conceptual model, i.e. the models do not represent the domain accurately. In [24], 6 antipatterns were presented, in general referring to instance-level constraints that OntoUML graphical notation cannot capture. The authors suggest to seek a solution in a form of OCL constraints, which in a certain degree, makes up for the lack of additional language constructs for instance-level constraints.

The tool support provided for OntoUML by the Alloy simulation does not require from the user to specify test cases; this, of course, makes validation more efficient. Nonetheless, it is also possible to further restrict the simulation by writing user-defined constraints. This leaves the modellers free to specify particular scenarios they want to analyse. This simulation approach for OntoUML is however limited mainly by the size of instances the tool is able to generate.

The DresdenOCL toolkit is able to provide direct OCL instance modelling and interpretation, but given that it is based on pure UML, it limits ontological expressive power rather significantly.

7 Conclusions and Future Work

In the paper, we dealt with instance-level modelling from the perspective of structural modelling. Instance-level modelling represents an important concept both for ensuring model quality and also for improving communication with non-technical domain experts.

We presented a traditional fact-oriented approaches – ORM – and a combination of OntoUML+Alloy+OCL supported by the Alloy analyzer. Both solutions have their pros and cons. The strength of ORM lies in a rich palette of constructs that enables very precise instance-level constraints specifications. We found the strength of OntoUML in the focus on ontologically well-formed models. Missing instance-level constructs may be provided using OCL, this, however, is a less elegant solution. By exemplification the meaning of the models, the Alloy tool helps the communication between modellers and domain experts.

Should we envisage possible future research for OntoUML+OCL+Alloy, we think that some of the OntoUML concepts should be revised from the perspective of instance-level modelling. The tool support is also under intensive development in both research groups mentioned in Acknowledgements. We also plan to start working on a two-way approach of instance and class-based modelling.

Acknowledgements. This paper was elaborated under the cooperation of:

– *The Centre for Conceptual Modelling* `http://ccm.fit.cvut.cz` supported by Faculty of Information Technologies, Czech Technical University and grant no. SGS13/099/OHK3/1T/18 of the Czech Technical University.
– *The Ontology and Conceptual Modeling Research Group (NEMO)* `http://nemo.inf.ufes.br` supported by FAPES (PRONEX grant 52272362).

References

1. Mylopoulos, J.: Conceptual modelling and telos. In: Conceptual Modeling, Databases, and Case: An Integrated View of Information Systems Development. (1992)
2. Gurr, C.: Effective diagrammatic communication: Syntactic, semantic and pragmatic issues. Journal of Visual Languages & Computing **10**(4) (August 1999) 317–342
3. Birkmeier, D., Overhage, S.: Is BPMN really first choice in joint architecture development? an empirical study on the usability of BPMN and UML activity diagrams for business users. In: Research into Practice Reality and Gaps. Volume 6093. Springer Berlin Heidelberg, Berlin, Heidelberg (2010) 119–134
4. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas: Manifesto for agile software development (2001)
5. Boehm, B., Basili, V.R.: Software defect reduction top 10 list. Computer **34**(1) (January 2001) 135137

6. Merunka, V.: Instance-level modeling and simulation using lambda-calculus and object-oriented environments. In: Enterprise and Organizational Modeling and Simulation. Volume 88. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 145–158
7. Hadar, I., Soffer, P.: Variations in conceptual modeling: Classification and ontological analysis. Journal of the Association for Information Systems **7**(8) (August 2006) 568–592 WOS:000246738800004.
8. Ullmann, S.: Semantics: an introduction to the science of meaning. Barnes & Noble (December 1978)
9. Halpin, T.: Fact-oriented modeling: Past, present and future. In: Conceptual Modelling in Information Systems Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 19–38
10. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Volume 015. University of Twente, Enschede (2005)
11. Guizzardi, G., Baio, F.A., Lopes, M., de Almeida Falbo, R.: The role of foundational ontologies for domain ontology engineering: An industrial case study in the domain of oil and gas exploration and production. International Journal of Information Systems Modeling and Design (IJISMD) **1**(2) (2010) 122
12. Carolo, F., Burlamaqui, L.: Improving Web Content Management with Semantic Technologies. SemTech, San Francisco (2011)
13. Silva, H.C.e., de Castro, R.d.C.C., Gomes, M.J.N., Garcia, A.S.: IT architecture from the service continuity perspective: Application of well-founded ontology in corporate environments. Journal of Information Security Research **3**(2) (2012) 4763
14. U. S. Department of Defense: Data Modeling Guide (DMG) for an Enterprise Logical Data Model
15. Group, O.M.: Semantic Information Model Federation (SIMF): Candidates and Gaps
16. Jackson, D.: Software Abstractions - Logic, Language and Analisys. Revised ed edn. The MIT Press, Cambridge, Massachusetts (2012)
17. Bordbar, B., Anastasakis, K.: UML2ALLOY: a tool for lightweight modelling of discrete event systems. In Guimares, N., Isaas, P.T., eds.: IADIS AC, IADIS (2005) 209216
18. Dennis, G., Chang, F.S.H., Jackson, D.: Modular verification of code with SAT. In: Proceedings of the 2006 international symposium on Software testing and analysis. ISSTA '06, New York, NY, USA, ACM (2006) 109120
19. Atinga, P., Krishna, A.: Verification of i* models using alloy. In: Information Systems Development. Springer New York (2011) 6374
20. Benevides, A.B., Guizzardi, G., Braga, B.F.B., Almeida, J.P.A.: Validating modal aspects of OntoUML conceptual models using automatically generated visual world structures. Journal of Universal Computer Science **16**(20) (November 2010) 29042933
21. OMG: Object constraint language, version 2.3.1 (January 2012)
22. Arlow, J., Neustadt, I.: UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition). Addison-Wesley Professional (2005)
23. Demuth, B.: DresdenOCL (March 2013)
24. Sales, T.P., Barcelos, P.P.F., Guizzardi, G.: Identification of semantic anti-patterns in ontology-driven conceptual modeling via visual simulation. In: 4th International Workshop on Ontology-Driven Information Systems (ODISE), together with the 7th International Conference on Formal Ontology in Information Systems (FOIS), Graz, Austria (2012)

Chapter 12

Towards OntoUML for Software Engineering: From Domain Ontology to Implementation Model

[202] Pergl, R.; Sales, T. P.; Rybola, Z. Towards OntoUML for Software Engineering: From Domain Ontology to Implementation Model. In Proceedings of MEDI 2013, volume 3rd, Amantea, Italy: Springer, Sept. 2013, ISBN 978-3-642-41365-0, pp. 249–263.

Towards OntoUML for Software Engineering: from Domain Ontology to Implementation Model

Robert Pergl, Tiago Prince Sales, Zdeněk Rybola

Department of Software Engineering,
Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic
{robert.pergl,zdenek.rybola}@fit.cvut.cz

Ontology and Conceptual Modeling Research Group (NEMO),
Computer Science Department,
Federal University of Esprito Santo, Brazil
tpsales@inf.ufes.br

Abstract. OntoUML is a promising method for ontological modelling. In this paper, we discuss its possible use for software engineering. We propose a method of transformation of an ontological model into a software-engineering object-oriented class model in UML and its instantiation. Our approach is based on the following best practices: pure object-oriented paradigm and approach of dividing state and identity as introduced in the Clojure programming language.

Key words: OntoUML, Sortals, UML, object-oriented modelling, models transformation

1 Introduction

Software engineering is a demanding discipline that deals with complex systems. The goal of software engineering is to ensure a quality software implementation of these complex systems. Various reports (e.g. Standish Group's) show that the success rate of software projects is far from satisfactory. We see the transformation of an ontological conceptual model into an implementation model as an engineering method that may contribute to better results and higher quality of the resulting software. We address the topic related to the *preservation of information between the successive software-engineering project phases* as discussed e.g. in [1]. We discuss the transformation of a structural conceptual model (in OntoUML) into a structural implementation model (in UML) during the design phase.

2 Goals and the Structure of the Paper

On one hand, we want to present our original method of transformation of ontologically well-founded conceptual models, written in OntoUML, into a pure object implementation model described by the UML class diagrams. To achieve that, we determine the following subgoals:

1. To develop a basic transformation of a conceptual OntoUML model into an implementation object-oriented model (section 4).
2. To develop a transformation of individual OntoUML entity types: We will limit ourselves to Sortal types in this paper (section 5).
3. To present a complete example of transformation of a sample model (section 6).

3 Methods and Materials

3.1 OntoUML

OntoUML is an ontology-based conceptual modelling language, initially proposed in Guizzardi's PhD Thesis [2] as a lightweight extension of UML (using the notion of Profiles). The language is based on the cognitive science achievements of understanding specifics of our perception and on modal logic and related mathematical foundations (sets and relations).

Unlike other extensions of UML, OntoUML does not build on the UML's ontologically vague "class" notion, rather it constitutes a complete system independent of the original UML elements. It uses some aspects (like classes), however, it omits a set of other problematic concepts (for instance aggregation and composition) and replaces them with its own ontologically correct concepts.

OntoUML is designed to comply with the *Unified Foundational Ontology* (UFO) and because of that, it provides expressive and precise constructs for modellers to capture a domain of interest. OntoUML addresses many problems in conceptual modelling, such as part-whole relations [3], and Roles and the counting problem [4].

The language has been successfully applied in different contexts. In [5], the authors developed an ontology using OntoUML for the domain of electrophysiology, which was used to provide semantic interoperability for medical protocols. Another application was for a domain of transport networks, in which it was applied to evaluate an ITU-T standard for transport networks [6].

OntoUML Sortals The notion of *Sortals* incorporated in OntoUML is the very same of the one in UFO, so in order to explain it, we must come back to the foundational ontology. One can understand UFO as a theory regarding individuals and universals. In conceptual modelling, universals are known as classes and individuals as objects, which instantiate these classes. Sortal universals are a special type of classes whose individuals follow the same identity principle, in contrast to Non-Sortal universals, which do not. Examples of Sortal universals

are Person, Student and Child. Examples of Non-Sortal universals are Customer (can be a Person or a Corporation – different kinds) and Insurable Item (can be a Car, a House or even a Person).

Identity principle is the feature that makes possible for one to count and distinguish individuals. The universal Person, for example, if understood from a social perspective, could define that its identity principle is one's social security number, whilst in a more biological perspective, it could be one's fingerprint. Some Sortal universals define the identity principle for their instances, whilst others just inherit it.

Another important ontological property of Sortals is *rigidity*. Some Sortals are rigid, which means that for every individual who instantiates the Sortal, they must always do so in every moment during its life cycle (e.g. Man, Company, Apple). On the other hand, some Sortals are anti-rigid, which means that an individual can instantiate it in a given moment and not do so in another one (e.g. Student, Employee, Child).

Kinds and Subkinds. In UFO there are three types of Sortals that define the identity principle for their instances (named Substance Sortal Universals): Kinds, Quantities and Collectives. Since the principle of identity of an individual must never change, these are also rigid types. In addition, every individual must have an (exactly one) identity principle, so all individuals must be an instance of exclusively one such universal. Subkinds represent another type of Sortals, which are rigid and they inherit the identity principle from other types. For that reason, they must always have a Substance Sortal as an ancestor. In this paper, we focus only on the Kinds and Subkinds Substance Sortals.

Roles. The Role Sortal defines an entity type of a Role an object plays when in a relation to other objects. Similar to Subkind, Role inherits the ontological identity from its Substance Sortal ancestor. It is an anti-rigid and *relationally dependent* concept. This means at the instance level that for an individual to play a given Role, i.e. to be a Role instance, it must participate in a relation with some other object. The anti-rigid property of Roles requires that we handle it dynamically in the implementation – the object can start to play the Role and stop playing the Role during its life.

Phases. In UFO, a Phase universal is a type whose instantiation is characterized by a change in an intrinsic property of an individual. Since it is out of the scope of this paper to discuss properties, it is sufficient to say that intrinsic properties are the opposite of relational properties. An intrinsic property can be structured, such as person's age, or unstructured, like one's headache.

Phases are anti-rigid and they inherit their identity principle from their Substance Sortal ancestor. An additional constraint on the Phase type is that it must be represented in partitions, i.e. Phases must belong to a disjoint and complete generalization set. An example of a Phase partition is phases of a person's life: A person can be a child, an adult or an elder. Each phase is characterized by a certain range of age and every person must be in exactly one of these phases.

According to the features of OntoUML mentioned above, we consider OntoUML very suitable for conceptual modelling as it can distinguish many various types of entities. Therefore, we also consider a tranformation of such a model into an implementation model very important to support the use of OntoUML in software engineering.

3.2 Object Model and UML

Object model started as a notion of programming paradigm in 1960s. It first appeared in the Simula programming language and it was nurtured in many following languages since then. Along the object-oriented programming (OOP), the object-oriented analysis (OOA) and the object-oriented design (OOD) together with object-oriented modelling (OOM) were established [7]. Today, object-oriented approaches dominate the software engineering and programming practice.

In this paper, we stick to pure object-oriented model which is exemplified e.g. in the Smalltalk programming language [8]. We utilize the following concepts:

Object – an entity that has encapsulated *inner state* in the form of **attributes** and reacts to **messages** sent by other objects by invoking **methods**, which are the only external representation of the object.

Class – a "template" that contains a structure of attributes and methods code. A class can spawn its **instances** – objects. In pure object model, class *is an object as well* and thus it may receive messages.

Constructor – a special *class* method for spawning instances (`new` in Smalltalk). By redefining the constructor, we may achieve special initialization of a new instance. This approach differs in many hybrid object languages (like C++), where the constructor is an *instance method* that is called *after* a new instance is created, thus having limited control over instance spawning.

Collection – a special object that can compose an unspecified (conceptually unlimited) number of instances. In pure object-oriented approach, the **members** of a collection may be of *heterogeneous* classes (which is unfortunately not true for hybrid languages like Java or C++).

We consider two basic relations between classes:

Inheritance – the *is-a*, or generalization-specialization relation. Here we deal with inheritance of classes, although inheritance of objects (without a class system) is also possible.

Aggregation [1] – the *part-of* relation between *objects*, where the aggregating object contains an aggregated object as its part. This is technically achieved by putting the object to an attribute. In the class diagram, aggregation is

[1] In UML, apart from aggregation, *composition* is discussed as another term for part-whole relations. For our purpose, we will use the term aggregation in its broader meaning including composition and also other flavours of part-whole relationships as discussed by Guizzardi in [9].

represented by a directed association (arrow) from the aggregating object's class to the aggregated object's class.

As for the notation, we will stick to the Unified Modeling Language (UML), a standardized general-purpose modelling language that was created for object-oriented software engineering. It includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. Although UML was designed to be a conceptual modelling language as well, it is better suited for implementation modelling, as it misses (and messes) crucial conceptual concepts [2]. We use a subset of *class diagram* notation for our purpose of the transformation of an OntoUML conceptual model into a UML implementation model here. For further information, see numerous literature available, we recommend [10].

3.3 The Concept of *State* and *Identity*

The concept of state and identity was introduced[2] in the Clojure programming language, a pure functional language coming from Lisp. However, its importance spans to conceptual modelling, as well. In our work we use it in the transformation to build completely dynamic object hierarchies that may be changed on-the-fly.

"Clojure introduces a philosophical and conceptual paradigm shift in its treatment of things. It takes the standard notion of a thing (an object or a variable) and decomposes it into two separate concepts - *state* and *identity*. Every thing has both a state and an identity. State is a value associated with an identity at a particular moment in time, whereas identity is the part of a thing that does not change, and creates the link between many different states at many different times. The value of each state is immutable and cannot change. Rather, change is modelled by the identity being updated to refer to a different state entirely." – [11].

In this approach the notion of *value* is very close to OntoUML's notion of *Quality Value*, where Quality is a sort of "identity" and Quality Value is its "state".

Hickey states that *state and identity are usually unified in object-oriented languages*, however, he also states that it does not have to be so.

In our approach, we apply the distinction of state and identity in pure object-oriented paradigm, as we discern *identity objects* and *value objects*. Identity objects are not allowed to hold any values, they just refer to the associated states. This enables us to build dynamic object hierarchies that may be changed on-the-fly. This could be especially useful when dealing with the history aspect of the data – for instance to capture the relations and attributes of an object in some history moment before changing its Phase or Role. We exemplify this approach in section 6.

[2] Here we do not want to go into philosophical roots of this concept, though we do not doubt that a discussion of this would be both deep and useful.

4 Transformation of OntoUML into Object Model

In this section, we present the foundations of a transformation. The transformation is well-known and very similar to the transformation of UML conceptual models. Therefore we emphasize the most important aspects of the tranformation into an object implementantion model. For details about specific Sortals transformation, see section 5.

An object model is very near to human thinking [12] and thus to OntoUML as well, so the transformation is rather straightforward (compared to – for example – a relational model). The object model contains less constructs than OntoUML. That means an OntoUML model thus needs to "collapse" into object-oriented concepts introduced in the previous subsection:

- Object (class) with attributes and methods,
- Object aggregation,
- Inheritance.

Various types of entities in OntoUML (`Kind`, `Subkind`, `Role`, ...) are transformed into object model **classes**. So, the discussion leads to the question how not to loose semantics during the transformation. Unfortunately, the space limitation does not allow us to discuss the issue in detail. We, however, redirect the interested reader to [1], where this issue is discussed.

4.1 Transformation of Attributes

An example of an OntoUML entity type with attributes is in Figure 1.

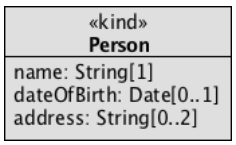


Fig. 1. An example of OntoUML entity type with attributes

The transformation of attributes is one-to-one with respect to the following:

- We may map types to a concrete object-oriented language or let them be general.
- Maximal multiplicities `0..1` of the attributes lead to instance variables.
- Maximal multiplicities `*` of the attributes lead to **collections**.
- For mandatory attributes with multiplicities `1..1`, it is necessary to ensure non-empty value of instance variable.
- For mandatory attributes with multiplicities `1..*`, it is necessary to ensure non-empty collection.

Ensuring Mandatory Attributes An object model contains no support for declarative specification of mandatory attributes. The mandatory attributes thus need to be handled *in methods*. The suggest the following two options:

1. "*Hard method*": Enforce a value insertion when creating or updating the instance. Prevent inserting empty value (*null* in most languages) in constructor and other methods by throwing an error.
2. "*Soft method*": Do not prevent inserting empty values but implement consistency checks when required – i.e. before persisting to a database.

4.2 Implementation of Associations

Association is a **bidirectional** relation between classes that has a name and **"multiplicity" values at both sides, i.e. 4 characteristics** (2x lower bound, 2x upper bound).

An example of OntoUML association is shown in Figure 2.

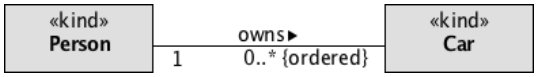


Fig. 2. An example of an OntoUML association

The following types of multiplicities are distinguished [7]:

1:1 – every instance of class A has (at maximum) one instance to which at most one instance of class B is associated.

1:N – every instance of class A may have one or more instances of class B associated (it corresponds to UML's `1..*` multiplicity).

M:N – an instance of class A may have more instances of class B associated and the instance of class B may have more instances of class A associated (corresponds to UML's `*..*` multiplicity).

The most direct implementation of an association is by *aggregation* (*part-of* relation between objects). Aggregation in object model is, however, a **unidirectional** relation. This is quite a complex issue and it may be solved by various approaches, as discussed e.g. in [13] or [14]. For the simplicity of concepts' explanations in this paper, we will limit ourselves to just *one-direction references* approach to transforming associations to object model – see Figure 3 for an example of 1:N-type association transformation. The direction of the aggregation is chosen according to primary navigation needs in this situation and we do not assume backward navigation.

As the aggregation leads to ordinary attributes, the issue of ensuring the minimal multiplicities of *1* at the opposite side of arrow – the target – is equivalent with ensuring the mandatory attributes described in section 4.1. Ensuring the mandatory presence of the instance on the arrow side – the source – is not trivial, as may be seen in [13], [14] or [15] and we will thus not discuss it here.

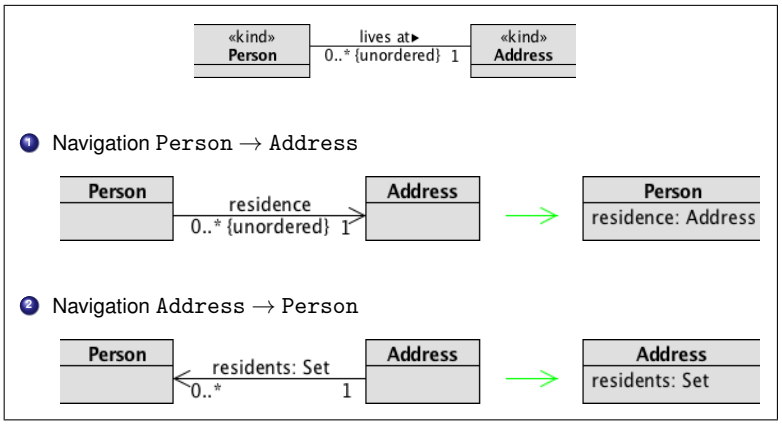


Fig. 3. Transformation of a 1:N bidirectional association into an unidirectional aggregation

5 OntoUML Sortals Transformation

This section discusses the main principles of transformations of individual Sortals into an object model classes. For each Sortal, we briefly and informally summarize its features that are important from the software-engineering point of view. We present the transformation into a class model in UML and we provide additional comments.

5.1 Kind and Subkind

The *Kind* type defines an object's identity and its attributes. According to our approach of dividing identity and state – see subsection 3.3, the `<<kind>>` entity type is transformed into two classes – one to hold the identity and the other to hold the state.

The *Subkind* type defines additional features of other Kinds and Subkinds, and thus forms a specialization hierarchy. It also makes the identity more accurate. Therefore, both the *identity* and the *state* parts have to be transformed along with the hierarchy. An example of the *Kind* and *Subkind* hierarchy is shown in Figure 4. For simplicity, we will not include the identity and state separating in the following parts describing the transformation of other OntoUML concepts.

5.2 Role

The most straightforward transformation of a Role type is to implement each `<<role>>` class in the OntoUML model by its own class in the UML model. However, because of the anti-rigidity, this class cannot be a specialization of its supertype from the OntoUML model because the specialization is rigid in UML

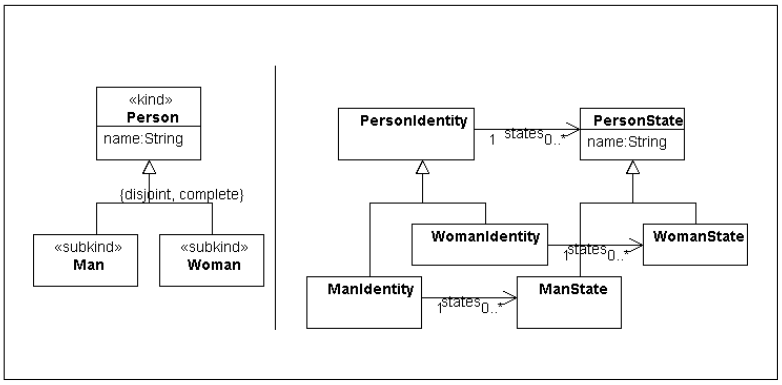


Fig. 4. Transformation of a Role type into a UML model

models and OO implementation languages – it cannot be changed during the life of the instance. Therefore, the generalization from the OntoUML model must be transformed into an *aggregation* that can be easily modified during the life of the instance without changing the identity provided by the associated rigid supertype. An example of such a transformation is shown in Figure 5.

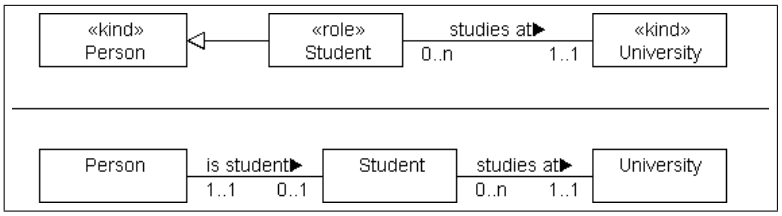


Fig. 5. Transformation of a Role type into a UML model

5.3 Phase

The transformation of the *Phase* entity type is similar to the transformation of Roles. Each `<<role>>` class in the OntoUML model is transformed into a UML class. To preserve the anti-rigidity of Phases in a UML model where the specialization and the generalization are rigid, the Phase partition hierarchy cannot be implemented by specialization. Instead, it is realized as an aggregation. However, compared to Roles, an entity may be in *just one* Phase of the Phase partition. We may denote this using XOR constraint, as shown in Figure 6, where an OntoUML model of various restaurant table states is transformed into a pure object implementation model in UML.

XOR constraint means that an instance of the *Person* type composes an instance of one of the phases in its `Phase` attribute. Additionally, the target

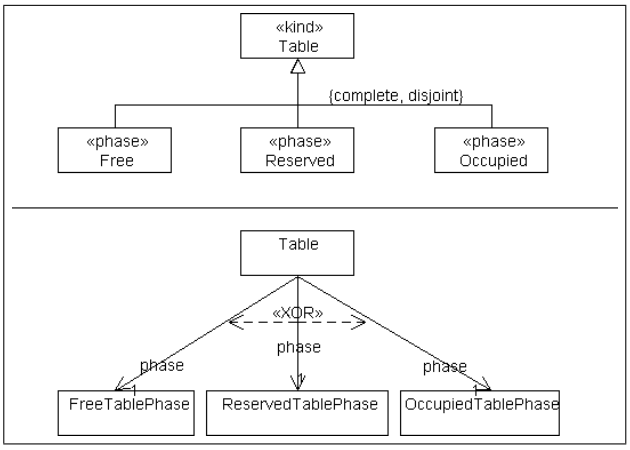


Fig. 6. Example of the Phase transformation

multiplicity constraint is *1*; that means that each instance of the *Person* type must contain an instance of just one of the phases.

In fact, the transformation is platform-dependent. In dynamically-typed object-oriented languages (Smalltalk, Python, ...) a programmer is able to put objects of various types into the same attribute. However, in static object-oriented programming languages (Java, C++, C#, ...), all attributes must define its type. To be able to put all the phases to the same attribute, an abstract supertype or an interface is necessary, as shown in Figure 7.

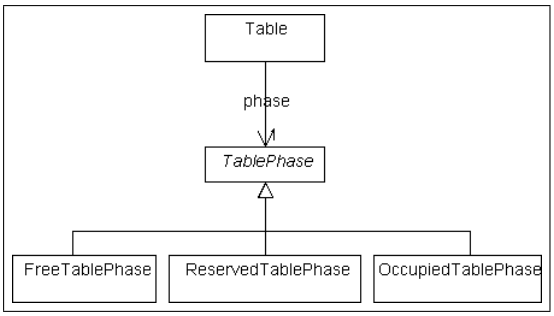


Fig. 7. Example of the Phase transformation for static object-oriented languages

6 Example of Transformation

In the example, we will include *history* information in the UML model, i.e. we track not just the current world, but all the worlds that happened to exist in the

past. As in [16], we "assume that the only thing specified in the domain ontology is how many times a certain substantial may play the same Role universal simultaneously, not throughout lifetime".

Figure 8 presents a sample model containing rigid and anti-rigid types. Figure 9 presents a transformed model according to the principles described above, where we need to:

1. Separate rigid and anti-rigid parts into distinct composed objects.
2. Separate identity and state (subsection 3.3).

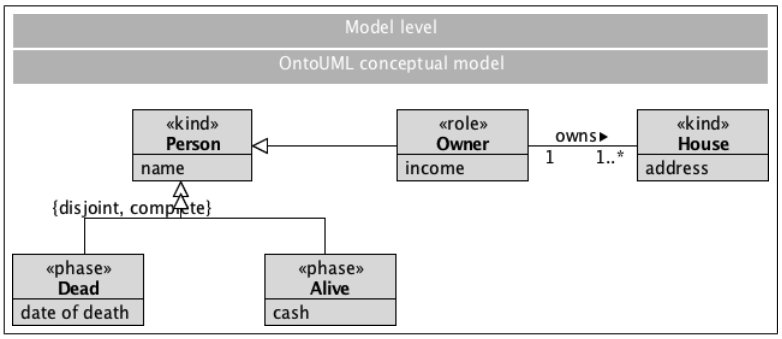


Fig. 8. Example of transformation – OntoUML conceptual model

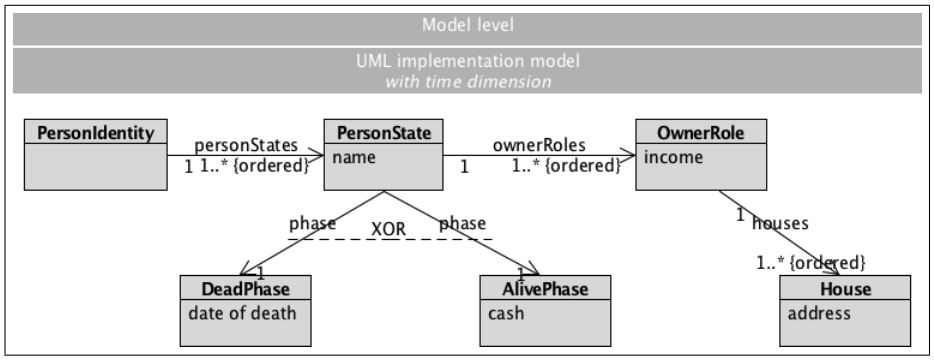


Fig. 9. Example of transformation – UML implementation model

As for the instance level, Figure 10 depicts a sample OntoUML instances in two different worlds. Having unlimited dynamic behaviour of entity types at instance-level OntoUML, we perceive just the resulting identities and their at-

tributes, while in the object model's instance world in Figure 11 the information is divided into the corresponding instances of classes[3].

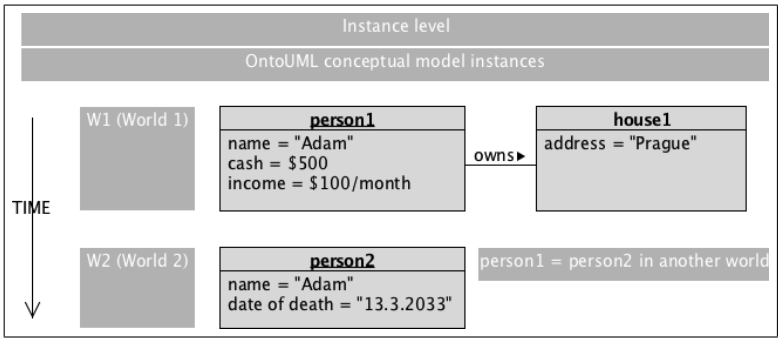


Fig. 10. Example of transformation – OntoUML instances example

7 Discussion and Related Work

The issue of conceptual modelling and domain ontologies have been discussed in many publications and papers. In [2], the author defines the OntoUML language based on Unified Foundational Ontology – UFO – for conceptual modelling without discussing its use in software engineering.

The use of OntoUML for software engineering is discussed in [16]. The author discusses the conceptual modelling at two levels – the ontological and the informational. He also suggests a transformation of an OntoUML model into an object-oriented UML implementation model. However, our approach differs in some aspects: In [16], the author presents transformation for object-oriented implementation model. We deal with a pure object implementation model and, therefore, there are some differences. These differences become rather crucial when dealing with anti-rigid types like Roles and Phases. Furthermore, the author transforms Role types from the OntoUML model into associations in the UML model. We, on the other hand, prefer transforming the Roles into separate UML classes in order to capture Role attributes. Consequently, we also use methods in the implementation to achieve the best quality object design. Moreover, we consider using aggregation of Phase classes and Role classes in the UML model to be more dynamic and, therefore, it conforms better to the pure object implementation. It also enables more flexible run-time manipulation of Roles and Phases. Additionally, we also deal with another OntoUML and UFO concept – Phases – that are not discussed in [16]. On the other hand, Carraretto deals with other entity types (Relators, Non-Sortals) that we do not cover here.

[3] We use a slightly simplified notation for collections here – *personStates* and *houses* collections are represented by a forked line leading to its members

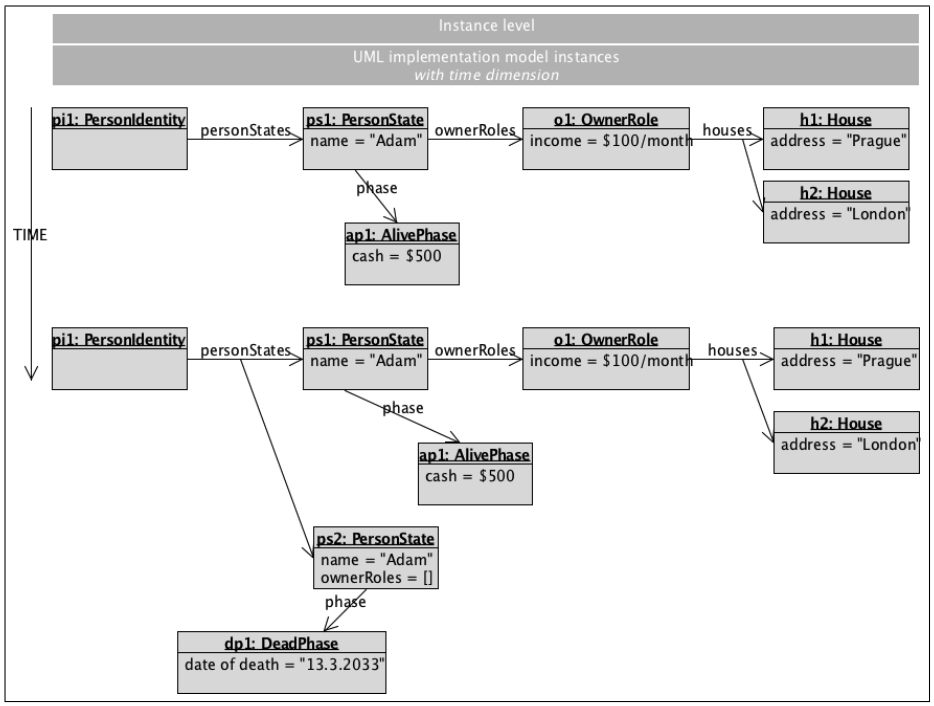


Fig. 11. Example of transformation – object instances example (UML)

The problem of object Roles and its transformation has also been discussed in many publications and papers – for instance in [17], [18] and [19] and others. The transformation of relationships has been also discussed in [15] in the context of relational databases and multiplicity constraints where the reader may find thorough discussion of the topics.

There are books and papers started to be written about the Clojure programming language (e.g. [20], [11], [21]). It is a pure functional language inspired by traditional Lisp and it provides several interesting ideas related to the topic of the paper. In this paper, we discussed the concept of separation of state and identity. However, the concept of *immutability* is also relevant to the topic of implementing OntoUML models – by implementing immutability of data structures, we get a complete separation of OntoUML worlds and so the mechanism shown in Figure 11 is maintained automatically by the interpreter.

8 Conclusions

Transformation of OntoUML (ontological) models into software engineering models is not a trivial issue, as we face limitations in current technologies. We discussed mostly a pure object-oriented technology here. It is perceived today as

the most advanced technology for implementing business information systems. Moreover, pure object-oriented paradigm is language-agnostic, as it may be (with more or less limitations) applied not only in pure object-oriented languages as Smalltalk, but also in hybrid ones like Java or C++.

With respect to OntoUML models, we see the following issues as crucial:

- Achieving maximum dynamic of systems – we prefer *aggregation* instead of static *inheritance*.
- Separating *state* from *identity* – here, we see Clojure's approach as very promising.

We see the following issues as important topics for future research:

- Generating automatic constraints checking for the transformed models for various paradigms and programming languages, as shown e.g. in [15].
- Transformations of OntoUML Non-Sortal entity types and other types (especially Relators and Part-Whole relations are very important in software engineering).
- Development of open, flexible and yet user-friendly and practical CASE tools supporting the transformations.

Acknowledgements. This paper was elaborated under the cooperation of:

- *The Centre for Conceptual Modelling* `http://ccm.fit.cvut.cz` supported by Faculty of Information Technologies, Czech Technical University and grant no. SGS13/099/OHK3/1T/18 of the Czech Technical University.
- *The Ontology and Conceptual Modeling Research Group (NEMO)* `http://nemo.inf.ufes.br` supported by FAPES (PRONEX grant #52272362)

References

1. Pícka, M., Pergl, R.: Gradual modeling of information system: Model of method expressed as transitions between concepts. In: ICEIS 2006 - 8th International Conference on Enterprise Information Systems, Proceedings. Volume ISAS. (2006) 538–541
2. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Volume 015. University of Twente, Enschede (2005)
3. Guizzardi, G.: The problem of transitivity of part-whole relations in conceptual modeling revisited, Amsterdam, The Netherlands (2009)
4. Guizzardi, G.: Agent roles, qua individuals and the counting problem. Software Engineering of Multi-Agent Systems (IV) (2006)
5. Goncalves, B., Guizzardi, G., Pereira Filho, J.G.: Using an ECG reference ontology for semantic interoperability of ECG data. Special Issue on Ontologies for Clinical and Translational Research (2011)
6. Barcelos, P.P.F., Guizzardi, G., Garcia, A.S., Monteiro, M.: Ontological evaluation of the ITU-T recommendation g.805. Volume 18., Cyprus, IEEE Press (2011)
7. Booch, G., Maksimchuk, R.A., Engel, M.W., Young, B.J., Conallen, J., Houston, K.A.: Object-Oriented Analysis and Design with Applications. 3 edn. Addison-Wesley Professional (April 2007)

8. Hunt, J.: Smalltalk and Object Orientation: An Introduction. 1st edition. edn. Springer (July 1997)
9. Guizzardi, G.: Representing collectives and their members in UML conceptual models: An ontological analysis. In: Advances in Conceptual Modeling: Applications and Challenges. Volume 6413. Springer-Verlag Berlin, Berlin (2010) 265–274 WOS:000289184200033.
10. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3 edn. Addison-Wesley Professional (September 2003)
11. VanderHart, L., Sierra, S.: Practical Clojure. 1 edn. Apress (June 2010)
12. Kay, A.C.: The early history of smalltalk. SIGPLAN Not. **28**(3) (March 1993) 6995
13. Dominik Gessenharter: Mapping the UML2 semantics of associations to a java code generation model. In: Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 813–827
14. Dominik Gessenharter: Implementing UML associations in java: a slim code pattern for a complex modeling concept. In: Proceedings of the Workshop on Relationships and Associations in Object-Oriented Languages. RAOOL '09, New York, NY, USA, ACM (2009) 1724
15. Zdeněk Rybola, Karel Richta: Transformation of special multiplicity constraints - comparison of possible realizations. In: FedCSIS 2012, Wroclaw, Poland (September 2012)
16. Roberto Carraretto: Separating Ontological and Informational Concerns: A Model-driven Approach for Conceptual Modeling. Master thesis, Federal University of Espirito Santo (2012)
17. Gottlob, G., Schrefl, M., Rck, B.: Extending object-oriented systems with roles. ACM Trans. Inf. Syst. **14**(3) (July 1996) 268296
18. Cabot, J., Ravents, R.: Conceptual modelling patterns for roles. In: Journal on Data Semantics V. Volume 3870. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 158–184
19. Bierman, G., Wren, A.: First-class relationships in an object-oriented language. In: ECOOP 2005 - Object-Oriented Programming. Volume 3586. Springer Berlin Heidelberg, Berlin, Heidelberg (2005) 262–286
20. R. Mark Volkmann: Clojure - functional programming for the JVM
21. Halloway, S.: Programming Clojure. 1 edn. Pragmatic Bookshelf (June 2009)

Chapter 13

Towards Formal Foundations for BORM ORD Validation and Simulation

[190] Podloucký, M.; Pergl, R. Towards Formal Foundations for BORM ORD Validation and Simulation. In Proceedings of the 16th International Conference on Enterprise Information Systems, SCITEPRESS - Science and and Technology Publications, 2014, ISBN 978-989-758-027-7, pp. 315–322.

Towards Formal Foundations for BORM ORD Validation and Simulation

Martin Podloucký, Robert Pergl
Department of Software Engineering
Faculty of Information Technology
Czech Technical University in Prague
Czech Republic
{martin.podloucky,robert.pergl}@fit.cvut.cz

Keywords: BORM, ORD, Process simulation, Process analysis, Formal foundations

Abstract: Business Object Relation Modelling (BORM) is a method for systems analysis and design that utilises an object oriented paradigm in combination with business process modelling. BORM's Object Relation Diagram (ORD) is successfully used in practice for object behaviour analysis (OBA). We, however, identified several flaws in the diagram's behaviour semantics. These occur mostly due to inconsistent and incomplete formal specification of the ORD behaviour. In this paper, we try to amend this gap by introducing so called input and output conditions, which we consider to be the most important first step towards a sound formal specification of the ORD.

1 INTRODUCTION

1.1 Motivation

Business Object Relation Modelling (BORM) is a complex method for systems analysis and design that utilises an object oriented paradigm in combination with business process modelling. It originated at the Loughborough University, UK in 1993. Successfull utilisations have been reported and published ever since, mostly in the area of IT and knowledge systems analysis and design (Knott et al., 2003), Object Behavior Analysis (Knott et al., 2000), (Merunka and Merunkova, 2013), Organization Modelling and Simulation (Brozek et al., 2010), ontological analysis (Pergl, 2011) and Business Intelligence (Merunka and Molhanec, 2011). Several other methods and techniques are based on the BORM method, such as FSM-Based Object-Oriented Organization Modeling and Simulation (Merunka, 2012), the C.C Language (Merunka et al., 2008) or a complexity estimation method called "BORM Points" (Struska and Merunka, 2007).

We agree with BORM's authors Knott, Merunka and Polak that there is a need for a simple, yet expressive tool for process modelling – and such a tool is BORM. In our experience, we can fully support (Knott et al., 2000) statement that it is a good approach to "start with a limited set of high level concepts which can subsequently be transformed into more software-oriented concepts necessary for the construction of a software oriented conceptual model", or – as other work on BORM suggests – into other types of artefacts and interpretations.

We have been using BORM successfully in practice for several years, as discussed in (Struska and Pergl, 2009), that we had an honour to present at ICEIS in 2009. Our professional focus is mostly on analysis and design of enterprise processes and behavioural analysis. Our practical experience led us to develop our own CASE tool to satisfy our needs in practical BORM usage. The first achievements were published in (Pergl and Tuma, 2012). After building a modelling tool that suited our needs, we started working on implementing *simulation* of BORM ORD (Object Relation Diagrams)[1], which is the core of BORM's behaviour aka process description (Knott et al., 2000). We were mostly inspired by Craft.CASE[2], for – as far as we know – there is

[1]This diagram is called "BOBA" ORD in (Knott et al., 2000), an abbreviation from "BORM Object Behaviour Analysis".

[2]http://craftcase.com

no other comparable tool for BORM diagrams simulation available today. Even though, one of BORM authors – Vojtech Merunka – gave a series of lectures on the Craft.CASE development, as witnessed in (Merunka, 2010), it seems, unfortunately, that there is no foundational paper that would explain the simulation semantics and rules in detail.

1.2 Goals

As advocated above, the main advantage of the BORM methodology and Object Relations Diagrams in particular is their great practical usefulness. On the other hand, we see one big disadvantage and it is a lack of sound formal foundations which would allow to clearly and precisely define the structure and semantics of ORD and other concepts related to BORM. Nowadays, many of such concepts are understood only intuitively. Therefore, the main goal of our work is to create sound formal foundations for BORM, which would not only help in understanding the semantics of BORM, but it will also help us to implement advanced software tools for this method.

The first results are presented in this paper, which addresses issues related to simulation and execution of the Object Relation Diagrams. The specific goals are:

- to thoroughly describe the semantics of BORM Object Relation Diagrams,
- to identify and discuss the main issues and ambiguities of the ORD semantics,
- to suggest an extension or modification of the ORD such that the above issues can be overcome and
- to start laying down the sound formal foundation for the ORD.

1.3 Structure of the paper

In section 2 we introduce BORM's Object Relation Diagram, being the focus of our study. We describe the basics of ORD together with minor changes to the meta-model we propose.

In section 3 we inspect the semantics of ORD and we try to deal with its basic ambiguities.

In section 4 we introduce the concept of input and output conditions which should resolve the described issues.

The rest of the paper follows a common structure: Discussion, Related work, Future work and Conclusion.

2 BUSINESS OBJECT RELATION MODELLING

This section introduces the *Business Object Relation Modelling (BORM) methodology*. We limit ourselves just to the Object Behaviour Analysis method (OBA)[3], its purpose, advantages and also details with respect to the issues we discuss here. In the following text, we abbreviate BORM OBA to "BOBA", as seen in (Knott et al., 2000).

Since BOBA generally studies processes, it is appropriate to explain our use of the term, in particular given the amount of definitions available in the literature. For the purposes of this work, we stick to the simple, practically-oriented definition provided by ISO 9000:2000:

Definition 1. Process is a set of interrelated or interacting activities that transforms inputs into outputs.

As the term activity has a specific (narrower) meaning in BOBA, we substitute ISO's term *activity* in the definition by *task* in this paper. Thus the term task will have a general informal meaning "something that needs to be done in order to accomplish a particular goal in a process".

2.1 Object Relation Diagrams

BORM methodology introduces *Object Relation Diagram (ORD)* to model processes and perform BOBA. Since in (Knott et al., 2000) only a very brief description of the ORD notation can be found, we start by a thorough description of the basic concepts of this modelling notation (Figure 1).

2.1.1 Participants, states, activities

The Object Relation Diagram is a graphical description of a process. It is essentially a collection of *participants* (depicted as grey rectangles), which are in turn collections of *states* (white rectangles) and *activities* (white ellipses). Each participant in ORD represents a person, an organization or a system partici-

[3]For a more thorough description of BORM itself, we refer the reader to the references.

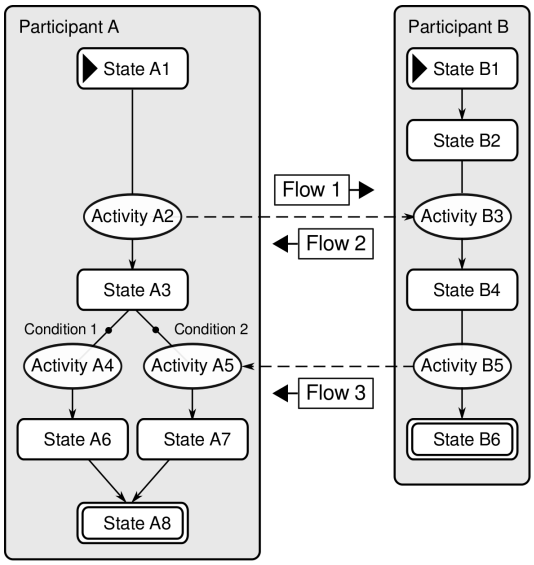


Figure 1: A sample Object Relation Diagram.

pating in the process. Participants follow the structure of the Mealy's machine (Mealy, 1955). States are thus the primary components of each participant. Each participant has exactly one start state (marked with black arrow) and at least one final state (drawn with double-line border) – this is **our first proposal** for ORD change, as the original syntax uses special symbols for a start state and a final state similar to UML Activity Models, while our concept is completely aligned with the definition of Mealy's machine, where the start state and end states are regular states.

States are connected by *transitions* which are represented as arrows. If two states *A* and *B* are connected by a transition, the participant being in state *A* can continue to state *B*.

On each transition, there may be an activity which describes what is happening when the participant transitions from one state to the other. The word "may" represents **our second proposal** for the ORD notation: In the original notation, the activities are required between the states, which is sometimes not wanted – analysts[4] then invent artificial activites like "no action", etc. Our proposal is nothing more than incorporating the notion of the ε-*transition* from the theory of finite state machines, i.e. a spontaneous transition from one state into the other.

The purpose of activities is twofold. First, following the semantics of the Mealy's machine, an activity represents an action which produces some kind of an output. Such an output may or may not be in fact tangible. In the end, an activity may simply be a task[5] that needs to be done by the participant in order to advance to the next state. The mere fact, that the task was done, is being considered as the output.

2.1.2 Communications and data flows

The second purpose of the activities is that they allow participants to communicate with each other. Activities can be connected by *communications* (drawn as horizontal dashed-line arrows). Communications represent channels for sending outputs of an activity of a participant to another participant. Such outputs are called *data flows*. Data flow is information or an artefact that is sent from a participant to another participant. The participant containing the activity with the outward communication arrow is always the *initiator* of the communication. Data flows sent by the initiator are called *input data flows*. In reaction, the receiving participant may send *output data flows* back to the initiator.

The original concept of communications in ORD presumes that both initiating and receiving participants must be in the initiating and the respective receiving activity at the same time in order for the communication to take place. We call such a communication *direct*. Howerver, in practical applications of ORD, we often find ourselves in the need for an indirect communications, as well. A direct communication models the situation where two interacting participants need to actually meet each other either personally, over a phone or using some other direct medium. On the other hand, if for example Bob writes an email to Alice, she does not have to wait at the other end to receive it. The e-mail waits for her until she opens it and Bob may in the meanwhile continue in his agenda. Thus, this represents **our third proposal**: the communication may be marked as *indirect*, which means that the initiator does not have to wait until the receiver arrives at his receiving activity and he may go right to the following state. The receiver, on the other hand, always needs to wait, until the respective data flow arrives.

2.1.3 Conditions

Transitions may be also restricted by *conditions*. If more than one transition comes out of a state, a condition may be placed on any number of the transi-

[4]Let us call the person doing the modelling an "analyst".

[5]We use the term *task* with the meaning explained in section 2.

tions. The participant may go forward along a transition only when its condition is met. Example of such a situation is shown in Figure 1. Conditions are used to express restrictions on decisions of participants and they are usually expressed in the natural language.

2.1.4 Other constructs

So far, we have described the basics of ORD semantics and graphical notation. There are also other, more advanced constructs in ORD. A state, for example, may contain a nested process; Conditions may be placed on communications as well. However, we do not deal with these contructs in this paper, since we identified that they bring serious complexity to the interpretation and simulation of processes.

2.2 ORD simulation

Apart from structural aspects of ORD, we need to discuss the behavioural aspects. In fact, simulation or execution of processes defined by ORD is the main challenge of our work. As already mentioned in the Introduction, there is no canonical definition of how the ORD process should by executed. As far as we know, the only implementation of ORD simulation is offered by the Craft.CASE modelling tool.

Figure 2 illustrates the first five steps of simulation of one simple participant, as performed by the Craft.CASE tool. When the participant is in a particular state, or it is performing a particular activity, such state or activity is highlighted with dark grey background. We say that such a state or activity is being *visited*. We see that the participant in the figure faces a decision at the start state *X* and chooses to proceed both of the possible ways simultaneously. We call such parallel ways *branches*. This illustration enables us to inspect the main issues with the simulation of processes defined by ORD.

3 REVISION OF ORD PROCESS BEHAVIOUR

ORD in the BORM methodology is a simple, yet a powerful way of describing and visualising business process. Its semantics can be easily described, especially to people with little technical knowledge in process modelling. That, of course, is a great advantage in the business environment. On the other hand, ORD still suffers from ambiguities in definitions and that, consequently, causes serious troubles especially when process simulation and analysis come to play.

3.1 Decision making and parallelism

Let us discuss the ambiguities and issues of the ORD behaviour that we mentioned above. Decision making and parallelism are the fundamental ones. There seems to be a lack of agreement on them. On the one hand, Knott, Merunka and Polak state in (Knott et al., 2000) that "BOBAs process model is strictly based on the theory of finite automata", namely on Mealy's machines. On the other hand, Brozek, Merunka and Merunkova explain in (Brozek et al., 2010) that "visual simulation of a business process is based on marked-graph Petri net" – but neither explain how these two different perspectives should merge together into a consistent and sound theoretical foundation and interpretation of the process specified by the ORD.

3.2 The simultaneity principle

The basic difference between Petri nets (Peterson, 1981) and Mealy's machines (Mealy, 1955) lies in the fact that Petri nets operate on the basis of massive parallelism, whereas Mealy's machines, when executed, always follow one simple path. The Craft.CASE tool obviously uses Petri nets to simulate processes as documented by Figure 2. However, this approach imposes a serious challenge of an ontologically correct interpretation of the notion of parallelism. From this perspective, we propose a notion of the *simultaneity principle*:

Principle 1. The **simultaneity principle** states that no participant can be split into multiple instances and thus perform several tasks in parallel.

This principle states that even though any participant may *be* in several states at once, no participant can actually *perform* several activities at once. The parallel branches in ORD have, therefore, ontologically this meaning – the activities belonging to different branches do not depend on each other. From that follows that such activities can be done *regardless of order*, which allows one to perform them *virtually* in parallel. Therefore, if a participant is required to do activities in parallel, the actual meaning is that it can choose to do them in any order desired, or switch between doing them, as wanted[6]. It is evident that this

[6]The situation may be compared to the preemptive mul-

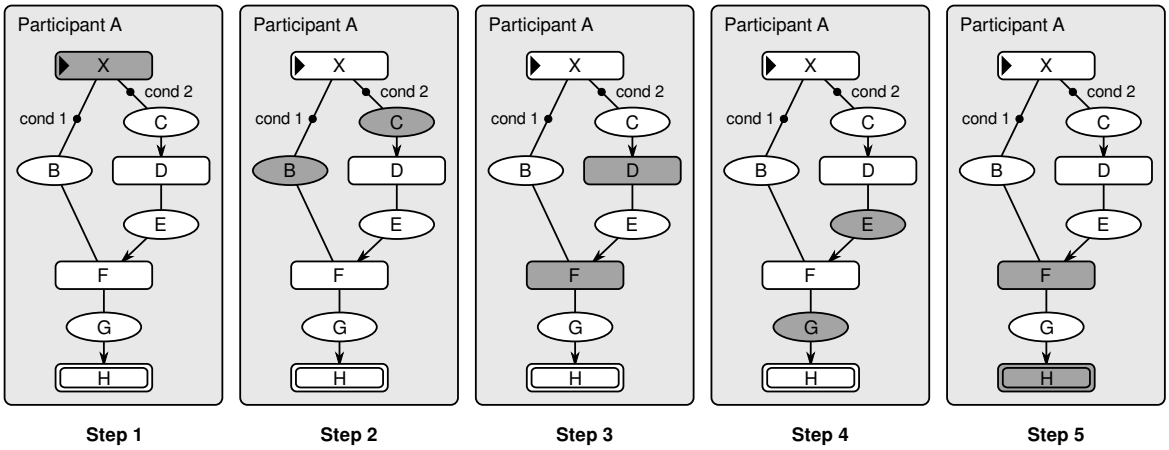


Figure 2: Simulation of a participant facing a decision.

concept imposes some constraints on the general behaviour of Petri nets, where multiple parallel tokens are moving independently through the structure of the net. The simultaneity principle is illustrated in Figure 2. If the participant A finds itself in the state X, it faces a decision where to go next. In the next step it appears to perform both the following activities at the same time which is, in fact, only a graphical illustration of the principle described above.

Furthermore, it is necessary to ontologically clarify what happens once the participant arrives to the state F. For example, if the participant arrives to F by the shorter of the two possible branches, should it wait in F until the other branch is completed as well? If so, what happens once the participant had chosen only one branch at the state X? In such a case the process falls into a deadlock. On the other hand, if we just follow the Petri net behaviour, no merge is performed and we get an ontologically extravagant situation as depicted in 2, where the participant actually arrives to F multiple times. This is in direct contradiction with the dependency principle.

The above issue could be solved simply by stating that F waits only for those branches, that had been actually chosen. Unfortunately, the nature of this issue seems to be deeper. When creating a process model, the analyst should be given a way to explicitly define which decisions are valid in a given state and which are not. For instance, executing several branches may not be possible in some situations in the reality. Instead, a process may require that there is precisely one of the possible branches that needs to be completed in order to advance further. Since simultaneous and exclusive choices are both valid in process definition and simulation, we come to the conclusion that neither Mealy's machines, nor Petri nets provide a sufficient formal description of the ORD process behaviour.

titasking of a computer processor.

3.3 The dependency principle

Before we introduce the second principle of ORD semantics, we first need to clarify some terminology. From the perspective of ORD, we talk about states and activities, states being the primary components of a participant in the process. When a transition is made from one state to another, an activity is performed and we say that the participant completed a *task*. To identify that task, we associate it with the state at which the participant has arrived. So the notions of task and state will be synonymous from our point of view.

Above, we already informally touched the notion of the *task dependency*, which is a very essential principle of process definition regardless of particular methodology. The terms "interrelated and interacting" in Definition 1 denote the fact that often several tasks have to be completed prior to completing another task. From now on, we refer to this principle as the *dependency principle*:

Principle 2. The **dependency principle** states that a task A may require other task to be completed before A can be completed.

The rules that determine on which tasks the task A depends, may be quite complex. Let us have a set of tasks $\{X, Y, Z\}$. For example, the task A may require a completion of *exactly two* tasks from this set. Thus, we need a sufficiently expressive system for specifying such dependency conditions – we introduce such a system utilising boolean algebra in the next section.

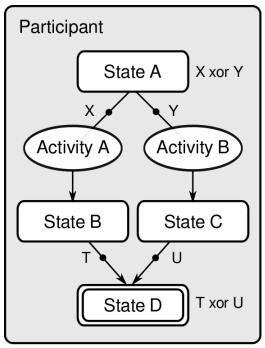


Figure 3: Sample input and output conditions.

4 INPUT/OUTPUT CONDITIONS

Having explained the main challenges, we may start formulating new formal foundations of the ORD. We start by introducing the concept of input and output conditions, which incorporates the dependency principle into the ORD and targets the formalisation of the simultaneity principle.

4.1 Input and output conditions

To be able to express the dependency principle in an ORD, we attach an input condition to each state:

Definition 2. Input condition of a state is a boolean expression whose variables are the transitions ending in that state. It specifies that the execution of the process cannot advance further from the given state until its input condition is met, i.e. until the corresponding boolean expression is evaluated as being true.

Similarly, each state also has an output condition.

Definition 3. Output condition is a boolean expression whose variables are the outgoing transitions from the given state. It specifies admitted combinations of branches into which the process execution may split itself from this state.

Figure 3 shows an example of input and output conditions allowing precisely two distinct paths through the participant's state graph. State *A* has an output condition which says that exactly one of two possible transitions may be chosen to continue forward. The state *D* has, in turn, an input condition saying, that exactly one branch is allowed to complete.

The input condition of a state is interpreted as follows: When several branches merge in one state, then this state waits for all of them to complete and only then evaluates its input condition. If the condition is met, the participant may advance to the next state, otherwise the process fails.

The output conditions, on the other hand, ensure that when the process flow splits into several branches, only the appropriate branches are allowed to be chosen, i.e. the branches that do not allow the process to fail – falling into deadlock. Therefore, input and output conditions provide a solution to addressing both the simultaneity and the dependency principles. Since each state waits for all the branches to complete before evaluating its input condition, it prevents the situation, where the participant would split into several independent instances, because one branch took more steps to complete. Moreover, each state now specifies exactly on what branches it depends and thus perfectly expresses the dependency principle.

5 Discussion

The solution of the issues with ORD process behaviour interpretation and simulation proposed here lies in the introduction of the input/output conditions of states. The solution has the following features:

1. Input/output condition are expressions in boolean logic and therefore general and unambiguous.
2. They address both the simultaneity and dependency principle.
3. No additional elements in the ORD are needed, so the diagrams remain clear and simple as was intended by the authors.

6 Related work

As we stated above, we are not aware of any systematic effort to build formal foundations of BOBA. Given that, our work is very likely quite novel for BORM. However, looking at model execution, simulations and behaviour analysis from a broader perspective, we may identify other attempts similar to ours – and at these, we want to look at now. There are generally two complementary approaches:

1. Start with a formal apparatus and build a practically applicable domain-oriented method and/or tool.
2. Start with a method used in practice and upgrade it into a simulation-able or an executable model.

Starting with the first type of approach, Brand's and Zafiropulo's Communicating Finite State Machines are an example. Their purpose was to design communication protocols (Brand and Zafiropulo, 1983). The authors took the finite state machines (FSM) theory and upgraded it consistently for modelling several together-bound FSMs. Another example of such an approach is Pattavita's and Trigila's proposal to combine the FSM with Petri nets for modelling communicating processes (Pattavina and Trigila, 1984). Another example is the Yasper tool for workflow modelling and analysis (van Hee et al., 2006); it is based on Petri nets enriched by several practical concepts from the domain of process analysis (hierarchies, choices, roles and others).

The second mentioned approach, i.e. to upgrade an existing method, is exemplified by our work. Kindred spirit to ours is Barjis: he proposed a method for developing executable models of business systems. Barjis' method is based on the DEMO method (Dietz, 2006). To make the static DEMO models executable, Barjis proposed a transformation into Petri nets (Barjis, 2007). His insight has been recently followed by, for instance, Vejrazkova and Meshkat (Vejrazkova and Meshkat, 2013).

We are also aware of similar approaches focused on standard "industry" notations UML and BPMN. In spite of general popularity of these notations, we do not deal with them, as they suffer from vagueness, ambiguities and ontological flaws, as mentioned by e.g. Silver in (Silver, 2011) or Dijkman et al. in (Dijkman et al., 2008). Guizzardi performed a deep analysis of BPMN suitability for expressing simulation models in (Guizzardi and Wagner, 2011) with quite discouraging results for researchers and practitioners focused on ontological soundness of modelling.

Though we could continue in compiling a list of similar approaches, our goal was just to document that a combination of a practical approach with a robust formal foundation leads to a new level of understanding of modelling methods, improving their expressiveness, power and, ultimately, their usefulness.

7 Future work

As implied by the title of the paper, our goal was just to make first steps towards sound formal foundations of BORM. The future work means to specify a complete formalism for ontologically sound execution and simulation of processes defined by ORD. This formalism should encompass the needed features of Mealy's machine and Petri nets, while at the same time not allowing ontologically extravagant situations.

In this paper, we omitted advanced ORD constructs (communication conditions and nested processes). These constructs should be also studied in the future work.

8 Conclusion

We described the syntax and semantics of BORM Object Behaviour Analysis (BOBA) – Object Relation Diagrams (ORD). We discussed the main issues and ambiguities of the ORD semantics with the respect to execution and simulation of processes defined by ORD. We proposed minor changes and enhancements for the model. Then, as the first step towards a sound formalisation of BOBA, we introduced the input and output conditions enhancement for states.

Our honest hope is that our contribution may be an inspiration for both BORM practitioners and formalists to join their forces to bring BOBA to a new level of expressive power and possibilities.

REFERENCES

Barjis, J. (2007). Developing executable models of business systems. Setubal. Insticc-Inst Syst Technologies Information Control & Communication.

Brand, D. and Zafiropulo, P. (1983). On communication finite-state machines. *Journal of the ACM*, 30(2):323–342.

Brozek, J., Merunka, V., and Merunkova, I. (2010). *Organization modeling and simulation using BORM approach*, volume 63 of *Lecture Notes in Business Information Processing*.

Dietz, J. L. G. (2006). *Enterprise ontology: theory and methodology*. Springer, Berlin; New York.

Dijkman, R. M., Dumas, M., and Ouyang, C. (2008). Semantics and analysis of business process models in bpmn. *Inf. Softw. Technol.*, 50(12):1281–1294.

Guizzardi, G. and Wagner, G. (2011). Can BPMN be used for making simulation models? *Lecture Notes in Business Information Processing*, 88 LNBIP:100–115. 00004.

Knott, R., Merunka, V., and Polak, J. (2000). Process modeling for object oriented analysis using BORM object behavioral analysis. In *4th International Conference on Requirements engineering, 2000. Proceedings*, pages 7–16.

Knott, R., Merunka, V., and Polak, J. (2003). The BORM methodology: a third-generation fully object-oriented methodology. *Knowledge-Based Systems*, 16(2):77–89.

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079.

Merunka, V. (2010). Object-oriented proces modeling and simulation – borm experience. *Trakia Journal of Sciences*, 8(3):71–87.

Merunka, V. (2012). FSM-Based object-oriented organization modeling and simulation. In Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M. J., Szyperski, C., Bajec, M., and Eder, J., editors, *Advanced Information Systems Engineering Workshops*, volume 112, pages 398–412. Springer Berlin Heidelberg.

Merunka, V. and Merunkova, I. (2013). Role of OBA approach in object-oriented process modelling and simulation. In Barjis, J., Gupta, A., and Meshkat, A., editors, *Enterprise and Organizational Modeling and Simulation*, volume 153 of *Lecture Notes in Business Information Processing*, pages 74–84. Springer Berlin Heidelberg.

Merunka, V. and Molhanec, M. (2011). BORM: agile modelling for business intelligence. In Rahman El Sheikh, A. A. and Alnoukari, M., editors, *Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications*. IGI Global.

Merunka, V., Nouza, O., and Broek, J. (2008). Automated model transformations using the C.C language. In Dietz, J., Albani, A., and Barjis, J., editors, *Advances in Enterprise Engineering I*, volume 10 of *Lecture Notes in Business Information Processing*, pages 137–151. Springer Berlin Heidelberg.

Pattavina, A. and Trigila, S. (1984). Combined use of finite-state machines and petri nets for modelling communicating processes. *Electronics Letters*, 20(22):915–916.

Pergl, R. (2011). Supporting enterprise IS modelling using ontological analysis. *Lecture Notes in Business Information Processing*, 88:130–144.

Pergl, R. and Tuma, J. (2012). OpenCASE – a tool for ontology-centred conceptual modelling. *Lecture Notes in Business Information Processing*, 112:511–518.

Peterson, J. L. (1981). *Petri net theory and the modeling of systems*. Prentice Hall.

Silver, B. (2011). *BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide: A structured approach for business process modeling and implementation using BPMN 2.0*. Cody-Cassidy Press.

Struska, Z. and Merunka, V. (2007). BORM points - new concept proposal of complexity estimation method. In Cardoso, J., Cordeiro, J., and Filipe, J., editors, *Proceedings of the ninth international conference on enterprise information systems*, pages 580–586. INSTICC.

Struska, Z. and Pergl, R. (2009). BORM-points: introduction and results of practical testing. *Lecture Notes in Business Information Processing*, 24:590–599.

van Hee, K., Oanea, O., Post, R., Somers, L., and van der Werf, J. (2006). Yasper: a tool for workflow modeling and analysis. In *Sixth International Conference on Application of Concurrency to System Design, 2006. ACSD 2006*, pages 279 –282.

Vejrazkova, Z. and Meshkat, A. (2013). Translating DEMO models into petri net. In *Enterprise and Organizational Modeling and Simulation*, volume 153. Springer Verlag Heidelberg.

Chapter 14

The Prefix Machine — a Formal Foundation for the BORM OR Diagrams Validation and Simulation

[200] Podloucký, M.; Pergl, R. The Prefix Machine – a Formal Foundation for the BORM OR Diagrams Validation and Simulation. In Enterprise and Organizational Modeling and Simulation, volume 191, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ISBN 978-3-662-44859-5, pp. 113–131.

The Prefix Machine – A Formal Foundation for the BORM OR Diagrams Validation and Simulation

Martin Podloucký and Robert Pergl( )

Department of Software Engineering, Faculty of Information Technology,
Czech Technical University, Prague, Czech Republic
{martin.podloucky,robert.pergl}@fit.cvut.cz

Abstract. Business Object Relation Modelling (BORM) is a method for systems analysis and design that utilises an object oriented paradigm in combination with business process modelling. BORM's Object Relation Diagram (ORD) is successfully used in practice for object behaviour analysis (OBA). OBA has found its firm place for visualisation and simulation of processes, however several ontological flaws were identified and there seems to be missing a strong formal foundation that would enable correct reasoning about the models. In this paper, we propose a sound formal foundation for BORM'S ORD. Based on this formal foundation (which we call "the prefix machine"), we get not only to a precise behaviour specification, but it also offers some interesting means of process analysis.

Keywords: Prefix machine · BORM · OR diagrams · Process simulation · Process analysis · Formal foundations

1 Introduction

1.1 Motivation

Business Object Relation Modelling (BORM) is a complex method for systems analysis and design that utilises an object oriented paradigm in combination with business process modelling. It is not probably necessary to introduce this method at EOMAS, so we will save precious space and dive into the problem directly. The diagram notation is not very complex and will be probably clear from the figures. Readers interested in more detail will find everything necessary in the references.

The authors of this paper have been using BORM successfully in practice for several years both in management consulting practice and research projects. Unfortunately, it seems there is no foundational paper that would explain the simulation semantics and rules in detail, which limits the use of BORM for mere diagramming and – inaccurate and ontologically not correct – simulations that are provided by the Craft.CASE tool [7]. We think that BORM has bigger potential for rigorous process analysis, as we intend to show in this paper.

1.2 Goals

In this paper we introduce a new formal model based on basic mathematical formalisms (sets, mappings, relations), graph theory and boolean algebra, that models the behaviour of BORM Object Relation Diagram (ORD). The model was designed with the following subgoals in mind:

- Simple, clear and sound formal model that would allow for formal foundations for conclusive BORM ORD behaviour interpretation.
- Having a formal description that serves as a specification (and probably also as a construction) for developing software for BORM ORD validations and simulations.
- The model should describe ideally all behaviour of BORM ORD that is commonly agreed upon.
- The model should not allow ontologically extravagant interpretations (i.e. interpretations that are not aligned with our perception of the process modelling domain).

1.3 Structure of the Paper

In Sect. 2 we briefly introduce BORM's Object Relation Diagram, being the focus of our study. In Sect. 3 we sum up the conclusions made in our previous paper [10] together with the current state of our work. In Sect. 4 we present our formal foundations of ORD. We begin with a simplification of the ORD model and we gradually define and describe the *prefix machine* and its execution model. In Sect. 5 we show an example to illustrate usage of the prefix machine. We also briefly describe process simulation and analysis based upon the prefix machine semantics.

The rest of the paper follows a common structure: Discussion, Related work, Conclusions and future work.

2 Object Relation Diagrams

Given that we set out to give a formal description of OR diagrams, we start by a very brief description of the basic concepts of this modelling notation together with minor changes to the meta-model we performed. A more thorough description of OR diagram notation can be found in our previous paper [10] and originally in the work introducing the object behavioural analysis [6].

Since processes are the major theme of our work, it is appropriate to explain our use of the term, in particular given the amount of definitions available in the literature. For the purposes of this work, we stick to the simple, practically-oriented definition provided by ISO 9000:2000:

Definition 1. Process is a set of interrelated or interacting activities that transforms inputs into outputs.

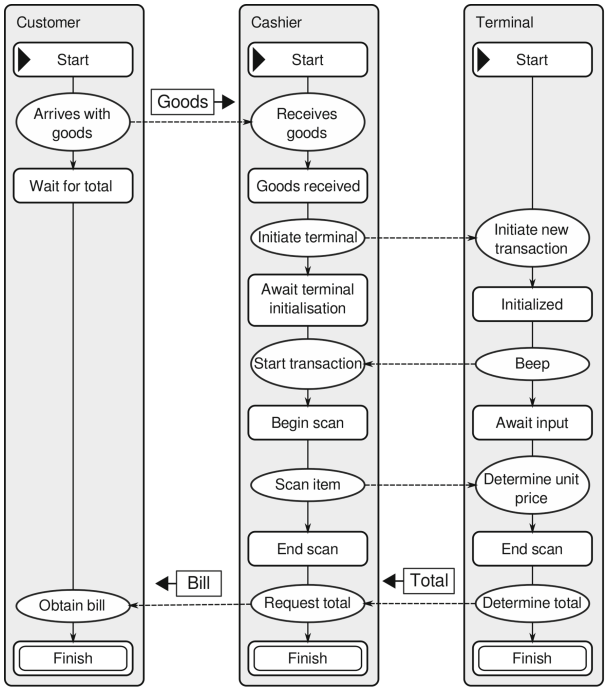


Fig. 1. A sample Object Relation Diagram. Inspired by [6].

To describe such a process, the BORM methodology uses the OR diagram. Figure 1 shows an ORD example taken from [6]. It describes a process of interaction among a customer, a cashier and a cash desk terminal.

The customer, cashier and terminal are so called *participants* of the process. Participants are basically state machines consisting of *states* – represented as rectangles – and transitions between them, represented as arrows. On each transition, there is an *activity*, represented as an ellipse. An activity represents an output of the process, which may simply be a task that needs to be done by the participant in order to advance to the next state. Activities from different participants can be connected by so called *communications*. These allow participants to communicate with each other. Communications are channels through which *data flows* can be sent.

For our purposes, we make one syntax change to the original BORM notation: We do not use extra symbols for start states and end states, rather we have switched to a notation more consistent with the definition of finite state machine: a start and a finite state are ordinary states distinguished only by an attribute. Graphically, we use a triangle symbol for a start state and a double line for an end state, as may be seen in Fig. 1 and the following ones.

3 Current State

In our previous paper [10], we started to explore the major issue of the BORM OR diagrams which is the lack of solid formal foundations. This issue becomes most apparent when one would like to actually execute the process defined by an OR diagram for the purpose of simulation or even orchestration. The simulation or execution of processes defined by ORD is therefore the main challenge of our work. As we have already mentioned in our previous work [10], there is no canonical definition of how the ORD process should by executed yet. Hence, we stand at the point where we need to investigate the semantics of the OR diagram and develop our own solution to this problem.

3.1 Simultaneity and Dependency Principles

When trying to identify the most fundamental principles underlying the execution of the OR diagram [10], we formulated the **dependency principle** and the **simultaneity principle**. As these principles have been already thoroughly discussed [10], we mention them here again only briefly.

Principle 1. The **simultaneity principle** states that no participant can in fact split itself into multiple instances and actually do several tasks in parallel.

This principle states that even though any participant may *be* in several states at once, no participant can actually *perform* several tasks at once. The parallel branches in ORD have, therefore, ontologically this meaning: The activities belonging to different branches do not depend on each other. From that follows that such activities can be done *regardless of order*, which allows one to perform them *virtually* in parallel. Therefore, if a participant is required to do activities in parallel, the actual meaning is that he can choose to do them in any order desired, or switch between doing them as wanted.

The statement that activities belonging to different branches do not *depend* on each other brings us to the *dependency principle*.

Principle 2. The **dependency principle** states that a task A may require other tasks to be completed before A itself can be completed.

The terms "interrelated" and "interacting" in Definition 1 imply the fact that often several tasks have to be completed prior to completing another task. The rules that determine on which tasks the task A depends may be quite complex. Let us have a set of tasks $\{X, Y, Z\}$. For example, the task A may require a completion of *exactly two* tasks from this set. Thus, we need a sufficiently expressive system for specifying such dependency conditions. We have already introduced such a system [10] which utilises boolean algebra and is called **input and output conditions**.

3.2 Input/output Conditions

To be able to express the dependency principle in an ORD, we propose to attach an input condition to each state:

Definition 2. Input condition of a state is a boolean expression whose variables are the transitions ending in that state. It specifies that the execution of the process cannot advance further from the given state until its input condition is met, i.e. until the corresponding boolean expression is evaluated as being true.

Similarly, we propose that each state also should have an output condition.

Definition 3. Output condition is a boolean expression whose variables are the outgoing transitions from the given state. It specifies admitted combinations of branches into which the process execution may split itself from this state.

Figure 2 shows an example of input and output conditions allowing precisely two distinct paths through the participant's state graph. State A has an output condition which says (using the classical boolean XOR operator) that exactly one of two possible transitions may be chosen to continue forward. The state D has, in turn, an input condition saying that exactly one branch is allowed to complete.

Having arrived at this point we have introduced all the necessary concepts from our previous work [10]. The main challenge now is to put these concepts on a solid basis and develop a formal framework for the execution of processes defined by OR diagrams. We need to precisely define the behavioural aspect of the OR diagram based on input/output conditions and the simultaneity principle. As explained in our paper [10], neither Mealy's machines nor Petri nets provide a suitable framework to begin with. Therefore, we need to develop our own formal machine, which would be capable of expressing the desired semantics of OR models regarding their simulation and execution. We will call this machine **the prefix machine**.

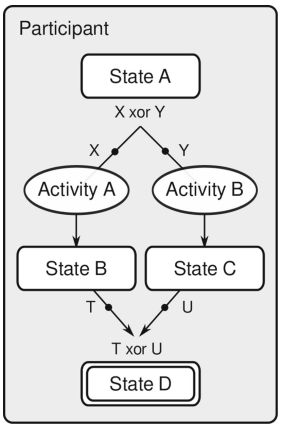


Fig. 2. Sample input and output conditions.

3.3 Simplified OR Model

As the current OR diagram does not have any formal model, we have to start from scratch and develop our own, maybe redefining some portions of the ORD along the way. Here starts our journey towards the prefix machine. In this section, we start a rather informal description of the prefix machine making ourselves ready for the more formal next section.

The prefix machine aims at formalizing a merge of both Mealy's machine and Petri net and overcomes the issues described above and in the paper [10]. The machine is based on directed graphs of states (similarly to the Mealy's machine), and its execution uses tokens (a trait typical for Petri nets). To describe the necessary concepts using the graph theory, we use the standard terminology [1].

The prefix machine is a somewhat simplified model aiming at capturing the essence of the "future" OR diagram while omitting unnecessary details and complexities from the current OR diagram. The prefix machine should then evolve into a formal basis upon which the future revised OR diagram should be based.

The prefix machine is basically structured as a directed graph. The correspondence between the current ORD and a directed graph is rather straightforward.

1. States correspond to vertices.
2. Transitions along with their respective activities correspond to edges.

We call the resulting digraph edges *transitions* as well, reducing, in essence, activities to transitions between states. At the same time, transitions represent dependencies between states, and thus express the dependency principle. For the sake of simplicity, we diverge from the current OR diagram structure a little by changing three important concepts.

1. We place communications between states instead of activities (recall that activities are no longer represented as vertices).
2. As it is convenient to have only one kind of edges in a directed graph, we replace communications in the model by transitions. We assume that in order for a communication to happen, both its source and target states have to be visited by their respective participants. This actually makes these two states dependent on each other, so we can *represent each communication with two opposite transitions.*
3. We do not allow cycles in the model, i.e. we discuss loop-less processes only.

None of the first two changes diminish the expressive power of the formal model in any way. The third change, on the other hand, certainly diminishes the expressive power and we are prepared to address this issue in our further research.

Notice, however, that by replacing communications with transitions, we already introduced cycles of length 2. A general directed graph can be transformed into an acyclic graph (to model a process) simply by contracting all strongly connected components into single vertices. Thus, after the aforementioned cycles removal, the source and target state of each communication collapses into a single *communication state.* Incidentally, such a result corresponds exactly to the intended meaning:

The act of communication between two participants in an OR diagram may happen just when both of them are visiting their respective communicating states.

Notice also that neither participants nor data flows are included in the simplified OR model. At this level of abstraction it is immaterial for us which task is executed by whom and what exactly is being sent along the communications lines.

4 The Prefix Machine

Now we are ready for the precise formal definition of the prefix machine. From now on, we use Greek letters such as φ, ψ to denote boolean expressions. We work rather extensively with them, with a special kind – **positive boolean expression**, in particular.

Definition 4. A boolean expression of at least one variable which evaluates to false when all of its variables are false, is called **positive**.

Let V be an arbitrary finite set. We denote by E_V the set of all positive boolean expressions whose variables are elements of V. Notice, that E_V does not contain formulas with no variables – the truth constant $\top$ and the false constant $\bot$. We deal with them separately. The rationale behind positive boolean expressions and constants will become clear later on, when we discuss the prefix machine execution. Now we have the tools for the long awaited formal definition of the prefix machine ready.

Definition 5. Prefix Machine G is a 5-tuple (S, T, K, e^-, f), where

- S is a set of digraph vertices representing **states**,
- $T = \{(x, y) \mid x, y \in S\}$ is a set of oriented edges representing **transitions**,
- $K \subseteq S$ is a set of **communication states**,
- $e^- : S \to E = E_T \cup \{\top\}$ is the mapping of states to their input conditions,
- $f \in S$ is the **terminal state**.

According to the above explanation, E_T is the set of all positive boolean expressions with variables drawn from T. The set of all possible input conditions E also includes the $\top$ constant. This is merely for convenience and it is justified further below.

To explain the notion of the terminal state, let us first consider the following notation. For a given state x we denote the number of its ingoing transitions by $\deg^-(x)$ and the number of its outgoing transitions by $\deg^+(x)$. The terminal state f is the one and only state for which $\deg^+(f) = 0$. On the other hand, there may be several states for which $\deg^-(x) = 0$.

Definition 6. A state $x \in S$ for which $\deg^-_G(x) = 0$ is called an **initial state**. Let us denote I_G the set of all initial states of G.

Definition 7. The condition e^-_f assigned to the terminal state is called the **terminal condition**.

Continuing the prefix machine definition, we add some further restrictions to it. For a state $x \in S$, let us denote its **outgoing transitions** as $t_x^1, \ldots, t_x^n$ and its **ingoing transitions** as $u_x^1, \ldots, u_x^m$.

The following statements hold:

$$\forall x \in S \; \exists \varphi(u_x^1, \ldots, u_x^n) \in E : \; e^-(x) = \varphi \tag{1}$$

$$\forall x \in K : \; \deg^+(x) = \deg^-(x) = 2 \tag{2}$$

$$\forall x \in K : \; e^-(x) = u_x^1 \wedge u_x^2 \tag{3}$$

Formula (1) says that there is an input condition for each state of G and all ingoing transitions of x are variables of that condition. This is where the constant $\top$ comes in handy since it is the only valid choice for the input conditions of the initial states which do not have any ingoing transitions. Formula (2) ensures that there are exactly two ingoing and two outgoing transitions for each communication state. This fact reflects the nature of a communication state: a communication state was created by merging two states from two different participants in the OR model. Thus, there must be exactly two paths going through each communication state, and each path has been taken by a different participant. Formula (3) ensures that both of those participants must be present in the communication state in order to advance further.

The definition of the prefix machine is now complete. Nevertheless, we need to define another important concept. Similarly to the mapping of the input conditions e^-, we define the mapping $e_G^+ : S \to E$ of **output conditions**.

However, this mapping is not a part of the prefix machine definition, since it is completely derived from the machine's structure and properties. The mapping is derived as follows:

- $e_G^+(f) = \top$
- $\forall x \in K : \; e_G^+(x) = t_x^1 \wedge t_x^2$
- $\forall x \in S \setminus \{K \cup \{f\}\} : \; e_G^+(x) = t_x^1 \vee \cdots \vee t_x^n$,

There are also several important facts implied by the definition of the prefix machine that are worth mentioning.

$$\forall x \in I_G : \; e_G^-(x) = \top \tag{4}$$

$$\forall x \in S : \; \deg_G^-(x) = 1 \Rightarrow e^-(x) = \mathrm{id}_u \tag{5}$$

Formula (4) says that when a state x is initial, there is no other possible choice for its input condition than the expression $\top$. Formula (5) says that when the state x has only one ingoing transition u, again, there is only one possible choice for its input condition – and that is a positive expression consisting of one and only one variable u. There is precisely one such expression for each variable u and it is the identity denoted by id_u.

4.1 Prefix Machine Examples

We can see two examples of prefix machines in Fig. 3. States are represented as rectangles and transitions as arrows. Let us first look at the machine G_1. We can see a sample input condition $u_1 \oplus u_2$ attached to the state z. Here we use the symbol $\oplus$ to denote logical non-equivalence (the same operator as XOR) and u_1, u_2 to denote ingoing transitions to the state z. Notice also the terminal state depicted as a crossed rectangle. Here we have omitted the obvious constant and identity input conditions which are attached to the initial states and states with only one ingoing transition. This prefix machine represents a participant, who needs to take one simple decision and the input condition ensures that they cannot choose to go both ways simultaneously.

The machine G_2 is a bit more complex. It actually represents two participants. This is clear from the fact that it contains two initial states. It contains one communication state depicted by a dashed rectangle which represents the situation where these two participants need to communicate with each other. Notice also the terminal condition attached to the final state, which says that both of these participants need to successfully finish. Examples of how such machines relate to OR diagrams and how exactly they are executed are presented in Sect. 5.

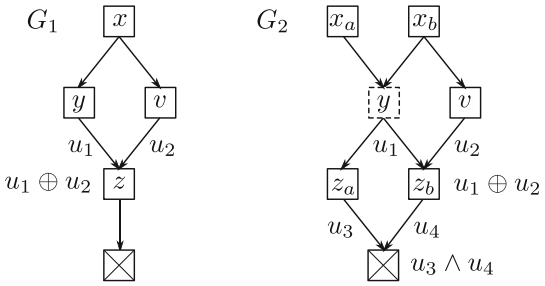


Fig. 3. Examples of prefix machines.

4.2 Prefix Machine Execution

Let us now proceed to the formal definition of a process simulation, as defined by the prefix machine. Let $G = (S, T, K, e^-, f)$ be a prefix machine. We put $\bar{S} = S \setminus \{f\}$.

Definition 8. State visitor is a mapping $g : \bar{S} \to \{\varnothing, \bullet, \circ\}$ with the following properties:

$$\forall x \in I_G : \; g(x) = \circ \tag{6}$$

$$\forall x \in \bar{S} : \; g(x) \neq \varnothing \Rightarrow \forall y((y, x) \in T \Rightarrow g(y) \neq \varnothing) \tag{7}$$

The purpose of the mapping g is to assign a **token** to each state of the machine with the exception of the terminal state. We distinguish between two types of tokens – *positive* (○) and *negative* (●).

Definition 9. We say that a state $x \in \bar{S}$ is **positively visited** or **having a positive token** if $g(x) = ○$.

Definition 10. We say that a state $x \in \bar{S}$ is **negatively visited** or **having a negative token** if $g(x) = ●$.

Definition 11. We say that a state $x \in \bar{S}$ is **unvisited** or **having no token** if $g(x) = \varnothing$.

We use the tokens as boolean variables with ○ representing the value *true* and ● representing *false*. By assigning a positive token to the state x, we express the situation where x's participant has already passed through x. A negative token means that the x's participant will never pass through x. No token means that in the current execution step, it has not been yet decided whether the state will ever be visited or not. Now we can define the notion of *configuration*.

Definition 12. The structure $C = (G, g)$ is called a **configuration** of G.

We can now interpret the above properties in the following way. Property (6) says that all initial states are always visited in all configurations of G. Property (7) ensures that when the state x is visited, all states on the path from the initial state to x are visited as well. The name *prefix* machine comes from this very property.

Sample prefix machine configurations can be seen on the left side of Fig. 4. White, grey and black circles represent unvisited, positively visited and negatively visited states respectively. Of course, there are many other configurations possible in this case. However, not all of them are interesting and useful here. Let us, therefore, define two special kinds of useful configurations.

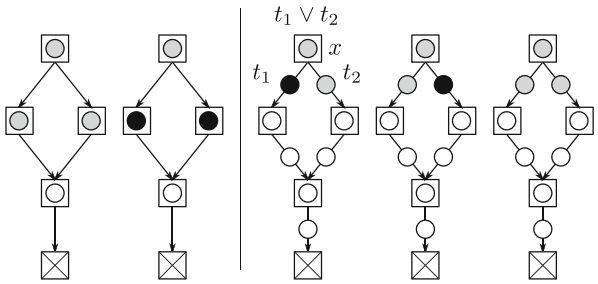


Fig. 4. Sample configurations of a simple prefix machine.

Definition 13. Configuration $C = (G, g)$ is called an **initial configuration** if $g(I_G) = \{\circledcirc\} \wedge g(\bar{S} \setminus I_G) = \{\varnothing\}$.

Definition 14. Configuration $C = (G, g)$ is called a **terminal configuration** if $g(\bar{S}) = \{\bullet, \circledcirc\}$.

From the above definitions we can see that each prefix machine G has precisely one initial configuration and possibly several terminal configurations. Now we would like to define relationships between different configurations of the prefix machine G. We do this by defining a mapping h called a *transition visitor*.

Definition 15. Mapping $h : T \to \{\varnothing, \bullet, \circledcirc\}$ is called a **transition visitor** if

$$\forall (x, y) \in T : \; h((x, y)) = \varnothing \Leftrightarrow g(x) = \varnothing \tag{8}$$

$$\forall x \in \bar{S} : \; g(x) \neq \varnothing \Rightarrow g(x) = e^{+}(x)[h] = e^{-}(x)[h] \tag{9}$$

This mapping is also required in order to evaluate the input and output conditions. We need to assign logical values to transitions, since they represent variables of those conditions. Thus, the symbol $\varphi[h]$ denotes the value of the condition φ under the mapping h (recall that we are using the tokens as logical variables). Now we can interpret the above formulas as follows. Formula (8) says that a transition is visited if and only if it is going out of a visited state. Formula (9) ensures that the token on the state x indeed corresponds to evaluation of both the input and output condition of x.

If there exists h that meets the properties (8) and (9) for a given configuration, it is called the **reachability witness**. Let's now denote $V_h = \{t \in T | h(t) \neq \varnothing\}$ the set of transitions visited by the reachability witness h. Now we are prepared for the following definitions.

Definition 16. Let C_1, C_2 be configurations of a prefix machine G. We say that the configuration C_2 is **reachable from** the configuration C_1 if C_2 is reachable by the witness h_2, C_1 is reachable by the witness h_1 and the following conditions hold

$$V_{h_1} \subseteq V_{h_2} \tag{10}$$

$$\forall x \in V_{h_1} \cap V_{h_2} : \; h_1(x) = h_2(x) \tag{11}$$

Definition 17. We say that the configuration of a prefix machine G is **reachable** if it is reachable from the initial configuration of G.

There may be more than one reachability witness for a configuration. The right side of Fig. 4 shows three possible witnesses for one initial configuration. This is allowed by the output condition of the state x.

Let's denote $\mathcal{R}_G$ the set of all reachable configurations of the machine G. By the properties of the $\subseteq$ relation we observe that the reachability relation on

$\mathcal{R}_G$ is reflexive, transitive and antisymmetric. This implies that the relation is a partial ordering of $\mathcal{R}_G$ with one minimal element – the initial configuration, and possibly several maximum elements – the terminal configurations. When we view this partial ordering as a directed graph we call this structure a **configuration graph**.

4.3 Process Simulation

Simulation of a process represented as prefix machine is done simply by traversing successive configurations of its configuration graph. Each path from the initial configuration to a terminal configuration is then called a **simulation course**. The configuration graph thus represents all possible simulation courses of a process and as such it can be used not only for process simulation, but also for process analysis.

5 Application and Examples

In the previous section, we developed a formal framework such, that it can serve as a base for sound definition of process simulation. This section presents two demonstrative examples illustrating the use of the framework to simulate and analyse processes in BORM. These examples also help to clarify the rather abstract and formal definitions presented in the previous section.

5.1 Example 1

As a first example, in Fig. 5, we can see a participant that needs to make a decision. This situation is modelled by the current OR diagram on one side and by the prefix machine on the other. The input condition at the terminal state z states that exactly one branch is allowed to be completed.

Let us look at all possible configurations of this machine pictured in Fig. 6. Configurations labelled as D are possible, but not reachable: there is no valid transition visitor satisfying all the required properties for any of them. On the

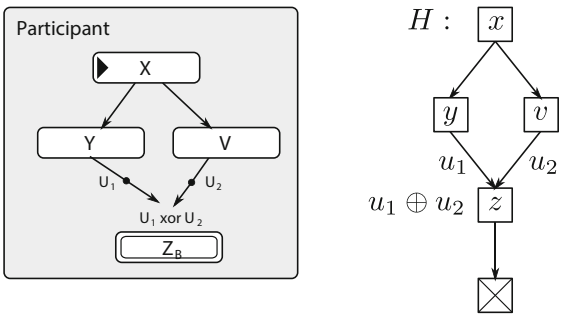


Fig. 5. A model of a simple decision participant using the prefix machine.

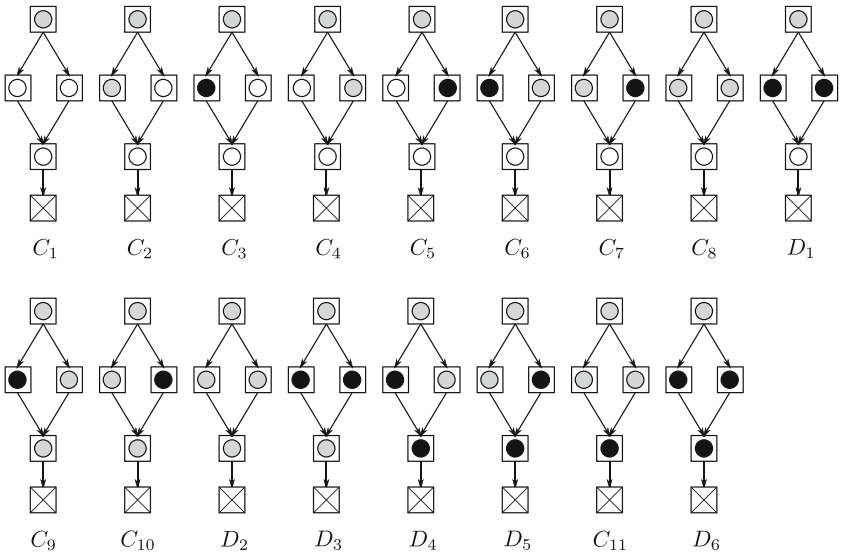


Fig. 6. All possible configurations of the machine H.

other hand, all the C configurations are reachable. In Fig. 7 we see all the reachable configurations with their respective transition visitors. Figure 8 then shows the resulting configuration graph.

Notice that the configuration C_{11} in Fig. 7 is terminal, but however, its joining state z is visited negatively and therefore the process arrives at the terminal state also negatively. Arriving at a terminal state may be interpreted as *achieving a specific goal*. Hence, the simulation course ending in configuration C_{11} did not reach the goal of the process. In the real world situation, we would say that the execution of the process has failed. The configuration graph in Fig. 8 even shows that when the simulation reaches the configuration C_8, it is doomed to eventually end up in C_{11} – and thus to fail. We use the *terminal condition* on the terminal state f to capture this notion formally. As the definition for the prefix machine says, all the transitions going into the terminal state are variables of the terminal condition. Logical values for these variables are provided by the transition visitor h. Therefore, the terminal condition can be evaluated only at a terminal configuration. In other words, the condition is evaluated at the end of the execution and states exactly which combinations of the terminal tokens represent a successfully completed execution of the process.

Now we have the tools to define the so called *failed* configurations.

Definition 18. We say that the **configuration is failed** when one of the following conditions holds.

– The configuration is terminal and the terminal condition evaluates to false.
– All paths from the configuration lead to failed configurations.

Finding failed configurations is very useful in a process execution analysis. Constructing a configuration graph and identifying the failed configurations

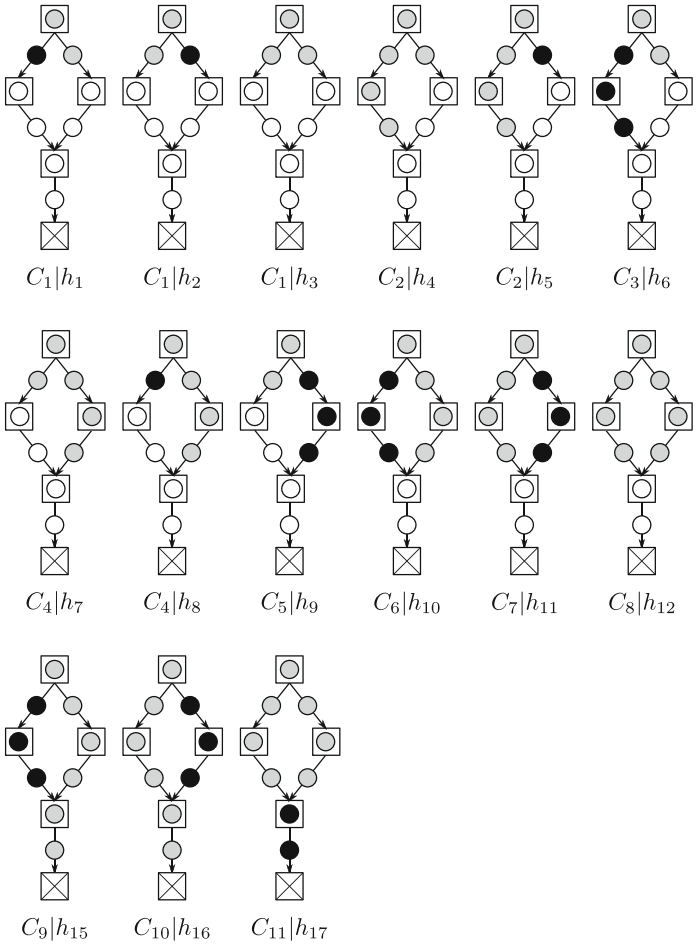


Fig. 7. All reachable configurations of the machine H with their respective transition visitors.

allows us to see valid scenarios of the process. When looking at the configuration graph in Fig. 8, we see all the failed configuration grayed out. Hence, we can easily see all the valid paths through the configuration graph.

Depending on our overall goals, we may consider the process successful even when just *some* of the participants reach terminal states (typically, the "customer" of the process). The purpose of the terminal condition in the prefix machine definition is precisely to allow such freedom in specification of the process goals. Since there is a straightforward mapping from the process machine states to participants, construction of the terminal condition can by done easily.

5.2 Example 2

The second example is a bit more complicated because it shows how communication states function. Figure 9 shows a model of two communicating participants.

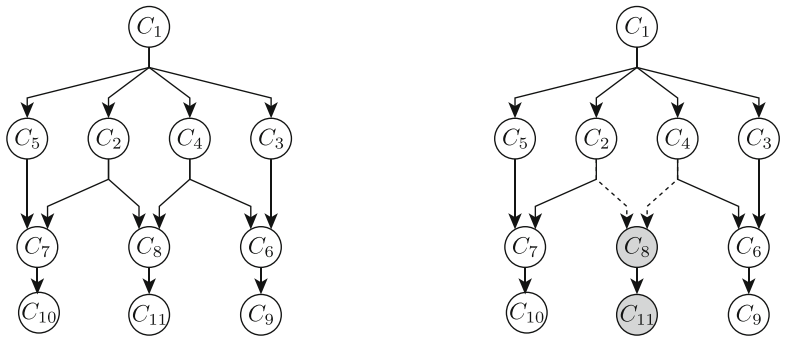


Fig. 8. The complete configuration graph of H (left). Failed configurations highlighted (right).

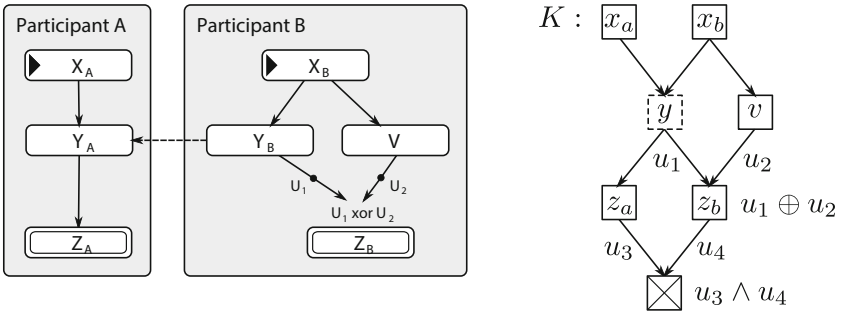


Fig. 9. A model of two communicating participants using the prefix machine.

Notice how the two communicating states from the OR diagram transform into one state in the prefix machine. As this example contains more states, its number of reachable configurations starts to grow large. Figure 10 tries to show all the 19 reachable configuration along with their 24 different transition visitors for an illustration purpose. It is now even more practical to look at the resulting configuration graph in Fig. 11.

You can notice an interesting thing there. It seems, that most of the reachable configuration are actually failed. This is due to the rather restrictive terminal condition seen in Fig. 9. This condition requires that both of the participants finish successfully. Notice, however, that when the participant B chooses to avoid communicating with the participant A by choosing to go to the other possible branch at state x_b, participant A cannot finish successfully. This is because A has no option to avoid communicating like participant B. This is a nice example of the process analysis framework. We have discovered that even though participant B seems to have a choice at state x_b, he actually has only one option in order for the whole process to succeed.

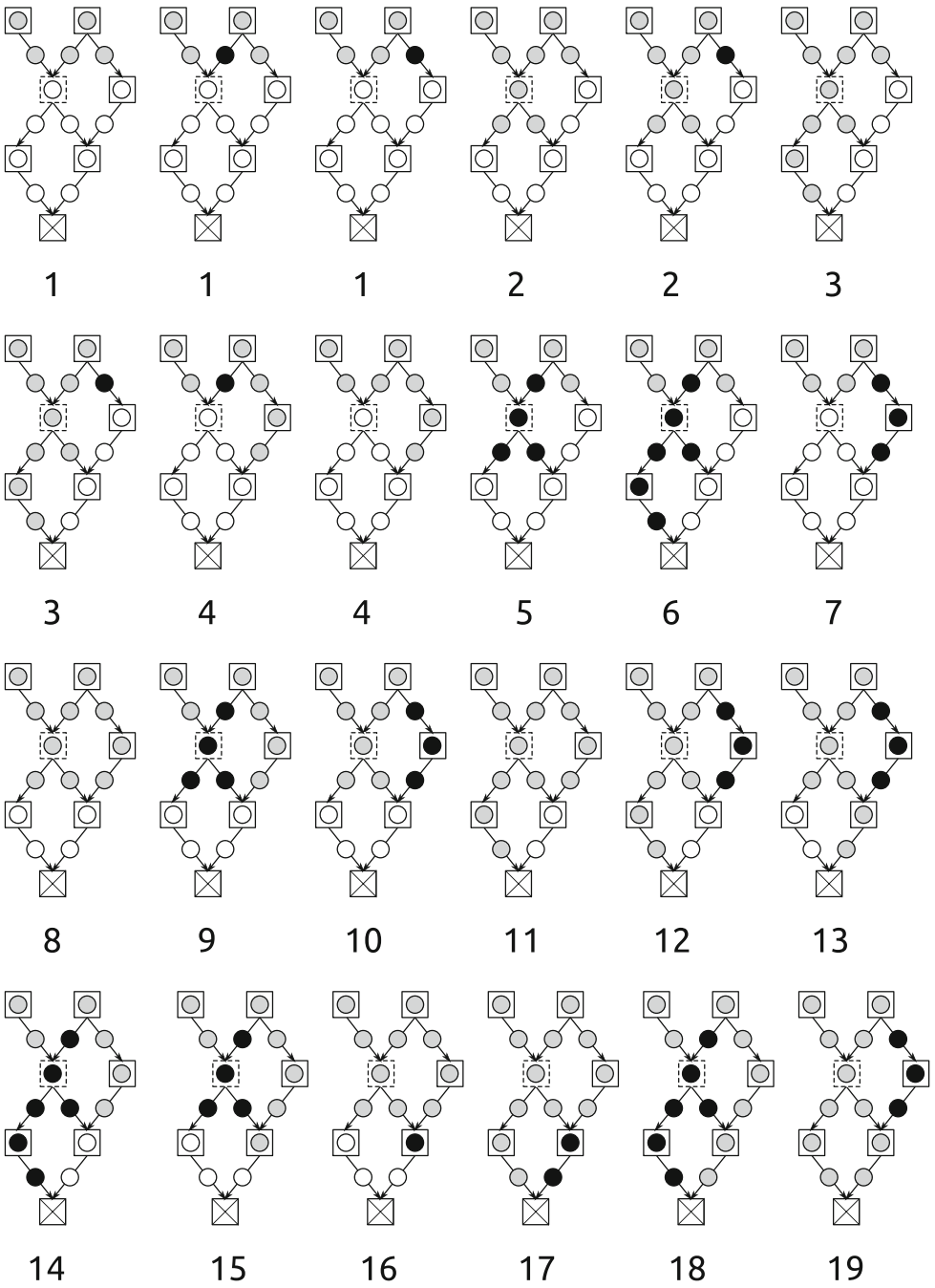


Fig. 10. All reachable configurations of the machine K with their respective transition visitors.

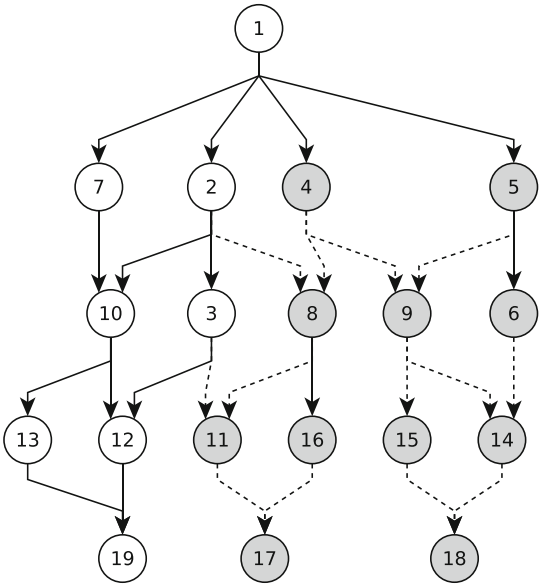


Fig. 11. The complete configuration graph of K with failed configurations highlighted.

6 Discussion

The previous section showed an example of how the prefix machine is used for simulation of BORM processes and how it can be applied to a process analysis.

The concept of failed configurations is a very useful tool in process analysis, especially if a suitable software were available. Such software tool may be used to algorithmically construct the configuration graph of a given process and, ultimately, to allow the user to inspect it closely or, simply, to learn more about the whole process. Such software, for example, could be used to identify all the courses doomed to fail and to illustrate them graphically; this, then, would identify in a neat way exactly those decisions in the process that lead to an inevitable failure.

It is not without interest that more accurate state output conditions (representing branching conditions) may be actually inferred from input conditions in the prefix machine. These inferred output conditions would allow only non-failing paths through the configuration graph when simulating the process. This opens another option for a process modelling tool which would enable the user to assign not only input, but also output conditions to every state. Then, an algorithm can be used to check whether those output user-specified conditions actually correspond to valid choices in the process by comparing them to the inferred output conditions.

7 Related Work

As we stated above, we are not aware of any systematic effort to build formal foundations of BORM OBA. Given that, our work is very likely quite novel for BORM. However, looking at model execution, simulations and behaviour analysis from a broader perspective, we may identify other attempts similar to ours – and at these, we want to look at now. There are generally two complementary approaches:

1. Start with a formal apparatus and build a practically applicable domain-oriented method and/or tool.
2. Start with a method used in practice and upgrade it into a simulation-able or an executable model.

Starting with the first type of approach, Brand's and Zafiropulo's Communicating Finite State Machines are an example. Their purpose was to design communication protocols [3]. The authors took the finite state machines (FSM) theory and upgraded it consistently for modelling several together-bound FSMs. Another example of such an approach is Pattavita's and Trigila's proposal to combine the FSM with Petri nets for modelling communicating processes [8]. Another example is the Yasper tool for workflow modelling and analysis [5]; it is based on Petri nets enriched by several practical concepts from the domain of process analysis (hierarchies, choices, roles and others).

The second mentioned approach, i.e. to upgrade an existing method is exemplified by our work. Kindred spirit to ours is Barjis: he proposed a method for developing executable models of business systems. Barjis' method is based on the DEMO method [4]. To make the static DEMO models executable, Barjis proposed a transformation into Petri nets [2]. His insight has been recently followed by, for instance, Vejrazkova and Meshkat [11].

8 Conclusion and Future Work

In this paper, we proposed a formal foundation for BORM Object Behaviour Analysis and Object Relation Diagrams (ORD). We call the resulting formalism "*the prefix machine*" (Sect. 4). This machine not only formally defines the behaviour of ORD, but it also enables the user to perform *automated process analysis* (Sect. 5) by finding valid and failed process scenarios.

With our prefix machine, we are now able to analyse the behaviour of OR diagrams, as we showed in the example and as we discussed above. We also proposed a couple of practical cases ready to implement in Sect. 5. However, we still miss a complete picture of the process in its discrete steps. So, to fill that gap, as a first step, we try to describe a *complete visualisation of the ORD behaviour* through the process. Another area for future work is to look at a formal model of cycles omitted by the prefix machine.

In order of the prefix machine to be useful and usable, we need a tool support. We plan to implement the prefix machine into the OpenCASE [9] tool.

Acknowledgements. This paper was written with the support of the SGS14 grant no. 103/OHK3/1T/18. The authors would also like to express their very great appreciation to Oskar Maxa for his insightful ideas and valuable contribution to this work.

References

1. Bang-Jensen, J., Gutin, G.Z.: Digraphs: Theory, Algorithms and Applications. Springer Monographs in Mathematics. Springer, London (2009)
2. Barjis, J.: Developing Executable Models of Business Systems. INSTICC - Institute for Systems and Technologies of Information, Control and Communication, Setubal (2007)
3. Brand, D., Zafiropulo, P.: On communication finite-state machines. J. ACM **30**(2), 323–342 (1983)
4. Dietz, J.L.G.: Enterprise Ontology: Theory and Methodology. Springer, Berlin (2006)
5. van Hee, K., Oanea, O., Post, R., Somers, L., van der Werf, J.: Yasper: a tool for workflow modeling and analysis. In: Sixth International Conference on Application of Concurrency to System Design, 2006. ACSD 2006, pp. 279–282, June 2006
6. Knott, R., Merunka, V., Polák, J.: Process modeling for object oriented analysis using BORM object behavioral analysis. In: 4th International Conference on Requirements Engineering, 2000. Proceedings, pp. 7–16 (2000)
7. Merunka, V.: Object-oriented process modeling and simulation - BORM experience. Trakia J. Sci. **8**(3), 71–87 (2010)
8. Pattavina, A., Trigila, S.: Combined use of finite-state machines and petri nets for modelling communicating processes. Electron. Lett. **20**(22), 915–916 (1984)
9. Pergl, R., Tůma, J.: `OpenCASE` – a tool for ontology-centred conceptual modelling. In: Bajec, M., Eder, J. (eds.) CAiSE Workshops 2012. LNBIP, vol. 112, pp. 511–518. Springer, Heidelberg (2012)
10. Podloucký, M., Pergl, R.: Towards formal foundation for the BORM OR diagrams validation and simulation. In: Proceedings of the 16th International Conference on Enterprise Information Systems, pp. 315–322 (2014)
11. Vejrazkova, Z., Meshkat, A.: Translating DEMO models into petri net. In: Barjis, J., Gupta, A., Meshkat, A. (eds.) EOMAS 2013. LNBIP, vol. 153, pp. 57–73. Springer, Heidelberg (2013)

Chapter 15

Revisiting the BORM OR Diagram Composition Pattern

[192] Podloucký, M.; Pergl, R.; Kroha, P. Revisiting the BORM OR Diagram Composition Pattern. In Enterprise and Organizational Modeling and Simulation, Lecture Notes in Business Information Processing, volume 231, Stockholm: Springer, 2015, pp. 102–113.

Revisiting the BORM OR Diagram Composition Pattern

Martin Podloucký( ), Robert Pergl, and Petr Kroha

Department of Software Engineering, Faculty of Information Technology,
Czech Technical University, Prague, Czech Republic
{martin.podloucky,robert.pergl,petr.kroha}@fit.cvut.cz

Abstract. This paper addresses the notion of process decomposition as a tool for managing process complexity in BORM Object Relation Diagram. It investigates the composition principle already present in ORD and shows it as ambiguous and mostly unsuitable for that purpose. Substantial changes to the original meta-model of ORD are proposed by introducing a new concept called tasks. The implications of introducing this new concept are then investigated, especially concerning decomposition of communications in a BORM process.

Keywords: Process composition · BORM · BOBA · OR diagrams

1 Introduction

The *Business Object Relation Modelling (BORM)* has been developed at the Loughborough University since 1993 [6] and later have became a practically-applied method with scientific interest, becoming a tradtitional topic of EOMAS [7–9, 11]. BORM is an elaborate method for systems analysis and design utilising object-oriented paradigm in combination with business process modelling. Since we are mostly interested in the business process modelling, the primary focus of our research is on the initial phase of the BORM called Object Behaviour Analysis (BOBA) [6].

1.1 Motivation

We have been using BOBA and other BPM methods successfully in practice for several years in management consulting practice, research projects and software development. Throughout the time, we began to greatly appreciate the BOBA approach to business process modelling, and we see many strengths and qualities in this approach. On the other hand, however, we also ran across several problems in the original BOBA, one of which is the concept of process decomposition.

It is probably not necessary to explain that real-life business processes are seldom clear and simple. In many cases, the processes are very complicated, and the resulting model is very difficult to comprehend. We believe that the most effective concept for managing such complexity in processes is the idea of composition. Unfortunately, the principle of composition in the BOBA's ORD (Object Relationship Diagram) is defined quite vaguely, and we see significant inconsistencies and limitations in its usage.

1.2 The Goal

The intention behind this paper is to address the issues of process composition in ORD. Our goal is to identify problematic spots in this original concept and eventually present a new approach and new definition of process composition which, once incorporated into ORD, would bring the BOBA to a next level of practical usefulness.

2 The BORM Method

In this paper, we use a version of ORD which is slightly different then the original concept described in [6]. In our previous work [12], we introduced some minor changes to the original syntax of the diagram to make it more compliant with the definition of Mealy's machine and to make it more useful in practice. We always point out these differences throughout the paper when in danger of confusion.

The original concept of BORM was inspired by experience in the area of software engineering. There is often a significant discord in the communication of ideas about particular problem domain between different stakeholders such as software developers and analysts on one hand, and business people on the other. This observation is supported be the original authors of BORM, who say that there is a problem to

> "... find a common language for the developers to express their understanding of the problem space that is both sufficiently rich for the developers to fully articulate their ideas, while also being comprehensible to users from all areas of discourse." [6]

We agree with this experience and, in unison with the authors, we believe BORM may be such a common language. As we explained in the motivation, we cherish the fundamental principles upon which BOBA is built. In our view, the most important of them is *simplicity*. The BOBA process model is based upon communicating finite-state machines which makes ORD especially clear in structure and appearance (see Fig. 1). This truly allows people with varying technical and business backgrounds to easily understand the process diagrams even at the very first sight. The properties that contribute most to the high comprehensibility of ORD are in our view:

1. a small set of simple and easy-to-understand building blocks with clear intentions behind them,
2. a straightforward depiction of participants of the process and
3. a very transparent depiction of communication amongst participants similar to that of UML sequence diagrams.

On the contrary, however, there are also several weaknesses in the original BOBA framework. Over the time we came into disagreement with some of the fundamental notions upon which the BOBA process diagram stands. Since we still believe that BOBA has a big potential especially regarding rigorous process analysis, our overall and long-term goal is to elevate BOBA to a next level by addressing its shortcomings and proposing appropriate enhancements to overcome them.

Our objections towards ORD are essentially twofold. Firstly, it is the lack of sound formal foundations of ORD meta-model and secondly, it is the unsatisfying definition of the composition.

2.1 The Formal Foundations

We agree completely with van der Aalst that solid formal foundations are very valuable since they allow for exact reasoning about the properties of the whole process, they do not leave any scope for ambiguity, and they increase the potential for analysis [1].

The original authors of BORM claim that ORD is based upon communicating finite-state machines (FSMs), namely Mealy's machines [3]. However, the proposed ORD behaviour diverges significantly from the execution semantics of the FSMs. This leaves a big scope for ambiguity in the execution and simulation of the modelled processes. In our previous work [11,12], we explored these problems to a great extent, and we observed that ORD *structure* is based upon Mealy's machines but the *behaviour* corresponds more to Petri nets. This led us to a formulation of a new formal model of ORD execution called the Prefix Machine [11], which is to some extent a combination of

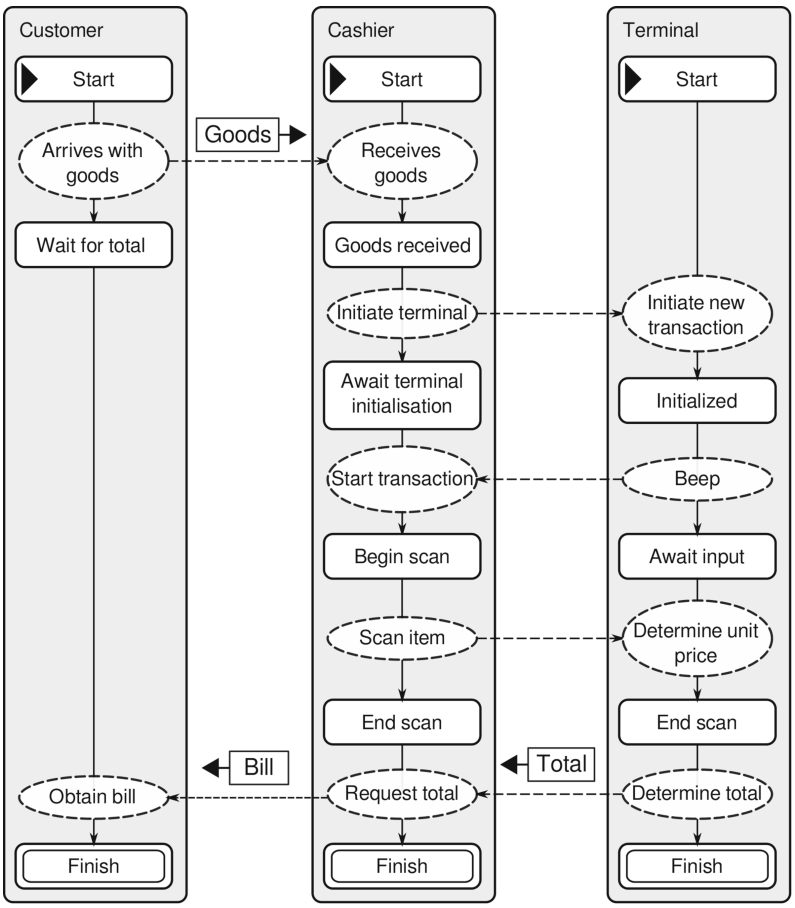


Fig. 1. A sample BORM Object Relation Diagram taken from [11]. Each participant is a Mealy finite-state machine. States are represented as rounded rectangles and activities as dashed ovals. Transistions are depicted as solid lines and communications as dashed lines.

Mealy's machines and Petri nets. That work introduced a precise formal grounding of ORD and BOBA. The relation between ORD, Mealy's machines, and Petri nets is discussed in our previous paper [12] in detail. Some open questions still remain, though. One of such open topics is process composition that we intend to explore in this paper.

2.2 The Process Composition

We believe that one of the most important tasks in process modelling is managing process complexity. Processes in the business reality are seldom simple and neat, the very opposite is quite often the case. As we explained in the beginning of this section, the primary aim of BORM and BOBA in particular is to serve as a communication language bridging the areas of discourse of software engineers and business stakeholders. As such, it strives for greatest simplicity and abstraction to suit the business people, yet still maintaining sufficient technical detail for the engineers. This is hard to achieve without a powerful system for managing the process complexity.

The human mind certainly has a limited capacity for processing visual information. A process diagram with hundreds of elements is difficult to comprehend even for a skilled person. A famous work by George Miller [10] in the field of cognitive psychology suggests that the number of objects an average human can hold in working memory is 7 ± 2. Having this argument in mind, we would like to have a well-defined concept of process composition in BOBA allowing us to decompose a process diagram into different levels of abstraction each of which contaning as few elements or concepts as possible.

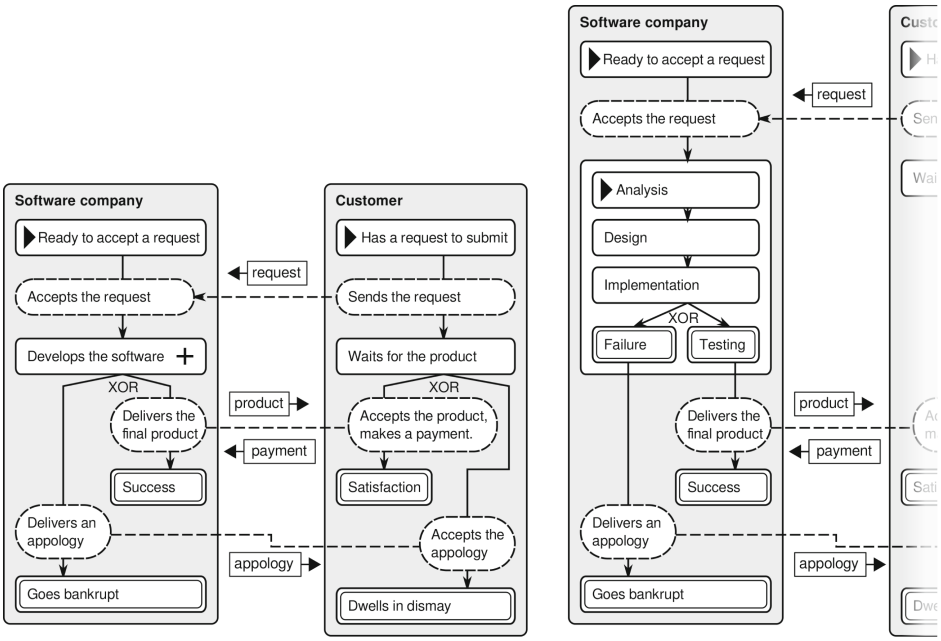
Object relation diagram in BOBA already has a construct to express process composition, however, we find it mostly unsuitable. In order to describe this construct and to support the further discussion, we are using a simplified version of the waterfall software development process as an illustrative example.

Figure 2 depicts this process in both collapsed and expanded form. Notice how the state `Develops the software` on the left side of the diagram is marked with a plus sign signalling that the state is composite. Composite states in ORD are used to encapsulate a sub-process and to hide it when a reader wants to abstract from too much detail. The sub-process can be revealed by expanding the composite state as seen on the right side of Fig. 2.

The waterfall example in Fig. 2 shows a standard way of using composition in BOBA. At the first sight, this concept may seem useful and straightforward. Though, under a more thorough look, serious doubts arise.

Imagine at first, that the software company is unable to create the product on its own and it needs an external supplier to develop a specialised library. As a result, the sub-process in the composite state now contains a communication, as depicted in Fig. 3.

Now the important question is, what to do with this communication when the composite state is collapsed. Since the ORD meta-model is based upon communicating finite-state machines, states cannot communicate, only the activities can. It seems, therefore, that states should not allow composition, and they should rather be atomic elements. Actually, this issue points to a more fundamental ontological problem in the BOBA method.



(a) State collapsed. (b) State expanded.

Fig. 2. Example of a simple composite state.

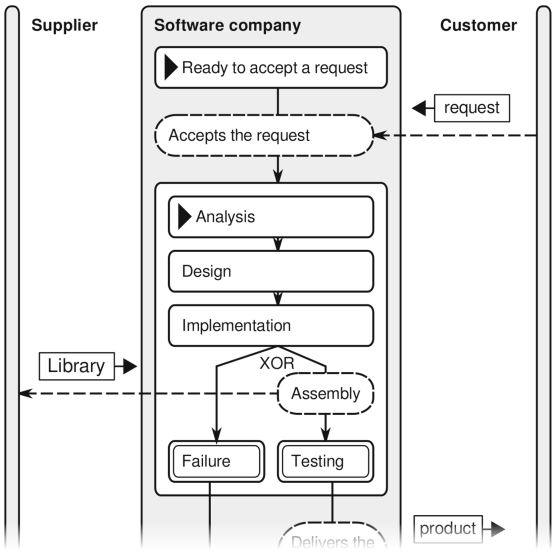


Fig. 3. Composite state with a communication.

Many a time, we see that processes modelled using BOBA contain states that represent some non-trivial work being done. Let us look again at Fig. 3 and at states such as `Analysis`, `Design` or `Testing`. In our view, this approach overloads the concept of a state too much. It is contradiction with the fundamental concepts of state machines and computation theory in general. There, a state is understood as a kind of milestone and not as a unit of work. By allowing composite states, BOBA itself goes against this very principle.

> To make BOBA compliant with the theory of FSM, we propose that states should be only milestones or named points in time. The only allowed activity in a state ought to be waiting. The only way how some actual work can be done in a process is by transitioning from one state to another.

2.3 New Composition in BOBA

From the above paragraphs, it seems that BOBA and ORD require some polishing and redefinition of some basic terms to make it more consistent. In fact, though, the problem of composition goes much deeper and revels broader area of ideas and problems to deal with.

In the following two sections, we elaborate on these challenges.

> First, we propose that ORD would benefit from more then a single principle of process decomposition. In fact, we see two orthogonal decomposition principles which can be incorporated into ORD to better manage the complexity of BOBA processes. We call them the *horizontal* and the *vertical* decomposition.

3 Vertical Decomposition

The term *vertical decomposition*[1] covers the principle of the composite states described in the previous section. Beware, however, that we had already dismissed the construct of composite states as contradictory. Yet, we still find the concept of vertical decomposition very useful. The question is, then, how to express this concept in a proper way.

The obvious first choice might be to use composite activities. As they represent transitions between states, they ought to represent the actual units of work in the process. This concept can be, however, dismissed just as promptly as the previous one using states. Let us look again at the right side of Fig. 2. The sub-process in the expanded state `Develops the software` has two mutually exclusive outcomes. These have to be mapped to corresponding distinct branches in the super-process. Yet, when we change the composite state to a composite activity, this is no longer possible since the control flow cannot branch at an activity[2].

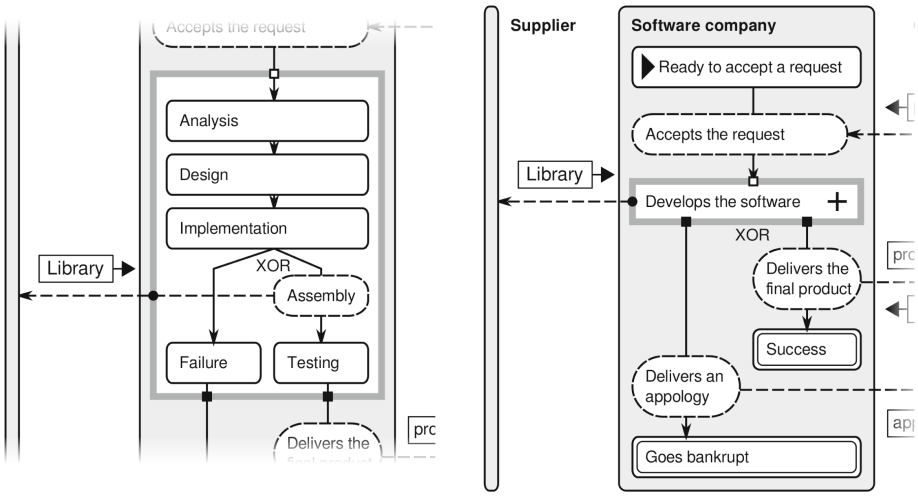
[1] The name of the notion comes from the visual appearance of ORD. When the composite states are collapsing, the diagram is shrinking vertically.

[2] The original version of ORD actually allows branching at an activity. The modified version we are working with now does not. It became necessary to restrict activities in such a way to get a better alignment with the definition of Mealy's machine [12].

Since only a state can be used to fork and join the control flow, we can use neither states nor activities as elements of composition without completely violating their fundamental semantics. It seems we need to keep states and activities atomic and introduce a completely new kind of element used as means of composition.

3.1 Tasks

We call the new composition element *a task*. Figure 4 shows the original composite state transformed into a task. In the diagram, we use a wide border to depict a task to distinguish it from the states and activities. As Fig. 4 shows, a task can be expanded and collapsed to show or hide the inner content.



(a) Task expanded. (b) Task collapsed.

Fig. 4. Example of a task.

A fundamental new concept coming with the tasks are so-called *ports*. They appear as symbols on the border of a task and they are used to define an *interface* of the task with the outer world. We distinguish two kinds of task interfaces, the *transition interface* and the *communication interface*.

The transition interface is used to specify the branching of the process flow. The interface uses the so-called *entry ports* (depicted as empty squares □) and *exit ports* (depicted as filled squares ■). Entry ports specify where a branch of the process may enter the task. Each task must have at least one entry port. When it has two entry ports for example, two distinct branches of the process need to enter the task. The number

of exit ports specifies how many branches exit from the task when it is completed (see Fig. 4). The task has no exit ports if the process terminates in it.

The communication interface is used to specify communications of a task and it is based upon exactly the same principles as the transition interface. The interface uses *inputs ports* (depicted as empty circles ○) and *output ports* (depicted as filled circles ●). Each input port represents a target of one communication and a task may have any number of such ports (including zero). Similarly, each task can have any number of output ports representing sources of communication.

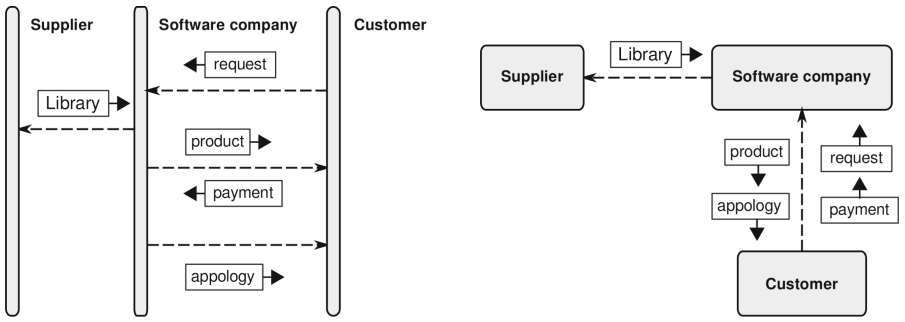
> Thanks to the ports, the transition and communication interface of a task remain exposed even when the task is collapsed. This is not the case in the original concept of composition using states or activities.

We can say that introducing our concept of tasks and interfaces in business process modelling corresponds to the concept of separation of interface and implementation in software engineering. This successful concept lead to the possibility of implementation evolution without changing the interface. In this case, the old, well-known method of stepwise decomposition may be viewed as corresponding.

4 Horizontal Decomposition

In fact, we have already been silently using the horizontal decomposition in ORD diagrams throughout the paper. See for example Fig. 3 where two of the participants are collapsed horizontally to omit unnecessary detail in their structure. Using this second kind of decomposition, any participant can be collapsed to a vertical line representing just its communication interface. This is another convenient way to hide a large amount of possibly insubstantial detail in the process diagram.

An interesting observation (though, without further elaboration in this paper) is the fact that we get a diagram very similar to the UML sequence diagram [4] (Fig. 5a) by



(a) A horizontally collapsed ORD reminds of a UML Sequence Diagram

(b) A totally collapsed ORD reminds of a UML Communication Diagram.

Fig. 5. UML-like diagrams of the waterfall model.

horizontally collapsing all of the participants. To bring this idea even further, we can at first collapse all the participants vertically as far as possible arriving at participants containing only the top-most task or tasks. Second, we can do complete horizontal collapse, eventually arriving at something similar to the UML Communication diagram [4] (Fig. 5b). In order to do this, however, we need to also collapse all the communications between participants to a single connection. This option opens a non-trivial discussion we touch in the next section.

Nevertheless, the horizontal decomposition does not seem to pose such challenges as the vertical one and it seems fairly consistent in its simplicity. Therefore, we leave it now and focus on the issue of vertical decomposition of communications.

5 The Challenge of Communication Decomposition

The above illustration suggests that the idea of collapsing communications to a single connection is rather straightforward. Yet, it is so only when we do not want to retain any structure of the communication interface among the respective participants. In this simple case, one cannot take one of the aggregated connections and connect it to an input port of some task, since the connection represents several single communications.

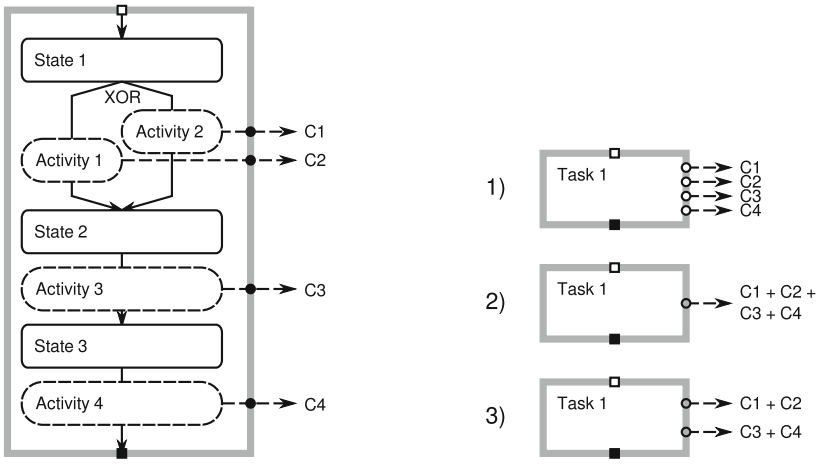
Eventually, we would like to be able to collapse the communication interface of a participant (or a task) as much as possible, while still retaining enough information about the structure of this interface, so it can be used to connect the participant to communication ports of other tasks in the process. Figure 6 shows a simple situation of this kind. The expanded version of the task on the left has four output communications. When this task is collapsed (on the right), there are several possibilities how to deal with the communications:

1. To leave them as they are. In that case, each communication can be connected to some input port right away, but we lose all the information about the structure of the interface.
2. To collapse all communications into one connection. This time, however, we cannot connect these connections to input ports right away since the only thing we know about the communication is its existence.
3. To group only parallel communications together. Yet, still are we left with aggregated connections which cannot be connected to input ports.

We argue that neither of the three proposed approaches is fully suitable. Some do not reflect the structure of the communication interface, whilst the others do not allow to connect the communications right away to input ports. The issue of communication decomposition seems to open a bigger topic, and we leave it to be investigated in a further research.

6 Summary and Discussion

Throughout the previous sections, we argued that the composition principle already present in ORD is not suitable as means for managing process complexity. The concepts of composite states or activities result in ambiguities and even contradictions



(a) Expanded task with several communications.

(b) The same task collapsed.

Fig. 6. Collapsing communications of a task.

(Sect. 2.2) with the basic notion of state in the finite-state machines. Having realised that, we strived to introduce a proper concept of composition into ORD. Our concept of tasks addresses the composition issues and brings flexible ways of composition resulting in new forms of ORD diagrams that resemble certain UML diagrams. However, the topic of composition is not totally solved, mostly due to challenges concerning decomposition of communications. The structure of communications of a task seems to contain an inherent complexity in it which is difficult to abstract from.

7 Related Work

Decomposition is an established concept in process-modelling methods and notations, as the goal of model comprehensibility is a natural driver, as we described in Sect. 2.2. Let us here briefly elaborate on most notable related efforts in other process-modelling approaches and their features.

Let us start with Yourdon's **Data-flow diagrams** (DFD) [14]. *Processes* are the holders of activities. Activities may be decomposed into a larger detail, i.e. DFD diagrams are generally recursive. We like the way decomposition is solved in DFDs, as it exhibits pure recursive behaviour and maintains consistency between the various decomposition levels: everything that goes into an activity on the level up needs to go into the decomposed diagram and so on. We adopt this approach of consistency handling, however it is impossible to maintain purely recursive composition in more complex approaches like BORM.

In the family of **Unified Modeling Language** (UML) diagrams [2,4], there are several behaviour-related diagrams that may be used to model certain aspects of processes. The basic ones are the Sequence Diagram and the Activity Diagram we already touched

above, the State Machine Diagram and the Communication Diagram. The `ref` operator is used in Sequence Diagram to reference an interaction defined in another diagram. In the Activity Diagram, an action can be decomposed into a subactivity. Input and output parametres may be specified, which can be related to our "ports" concept. The State Machine Diagram allows nested states, however, we perceive a serious flaw in the fact that the nested state may have a transition to the state in the level above, which violates separation of levels and kills recursivity. The Communication Diagram (called the Collaboration Diagram in UML 1.x) enables the decomposition of message flows, thus expressing the routine-subroutine relations in an algorithm.

Business Process Modelling Notation (BPMN) [13] offers a quite rich assortment of decomposition of activities:

- A *Task*, a basic unit of work may be decomposed into a sub-process.
- A *Transaction* is a set of activities that logically belong together; it might follow a specified transaction protocol.
- An *Event Sub-Process* is placed into a Process or Sub-Process. It is activated when its start event gets triggered and can interrupt the higher level process context or run in parallel (non-interrupting) depending on the start event.
- A *Call Activity* is a wrapper for a globally defined Task or Process reused in the current Process.

Moreover, various activity markers may be used to refine the behaviour of decomposed activities: the Loop Marker, the Parallel Marker, the Sequential Marker, the Ad Hoc Marker. The broad offer of decomposition concepts in BPMN may be a benefit for executable BPMN models, however, we find them too complex and hardly comprehensible for business users.

The last related work, we mention here, is the **Hierarchical Coloured Petri Net** (HCPN) extension [5], which brings the concept of *modules* into the standard Petri Net. Modules have ports similar to our approach, apart from input and output ports, input-output ports are supported. HCPN introduces also a new type of model, the Hierarchical Protocol Model, which shows the decomposition relations. The decomposition in HCPN is clean and recursive, which is possible mostly due to the simple nature of Petri Net concepts.

8 Conclusion and Future Work

We proposed a new model of decomposition for the BORM OR Diagrams. The new model addresses ontological and formal flaws of the original "naive" way of decomposition and offers interesting new concepts like vertical and horizontal decomposition. At the same time, new challenges arise, mostly in the topic of communication (de)composition, which would support further abstraction and detail hiding. Also, by introducing the concept of tasks, we substantially changed the structure of ORD and its execution semantics which needs to be reflected in its formal meta-model. The Prefix Machine mentioned at the beginning of this paper needs to be revisited to accommodate these new concepts.

Last but not least, the new decomposition concepts need thorough testing in practice to show possible ontological or practical flaws in special situations.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: a survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
2. Arlow, J., Neustadt, I.: UML 2.0 and The Unified Process: Practical Object-Oriented Analysis and Design, 2nd edn. Addison-Wesley Professional, Boston (2005)
3. Brožek, J., Merunka, V., Merunková, I.: Organization modeling and simulation using BORM approach. In: Barjis, J. (ed.) EOMAS 2010. LNBIP, vol. 63, pp. 27–40. Springer, Heidelberg (2010)
4. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, Reading (2003)
5. Jensen, K., Kristensen, L.M.: Coloured Petri Nets. Springer, Berlin (2009)
6. Knott, R., Merunka, V., Polák, J.: Process Modeling for Object Oriented Analysis using BORM Object Behavioral Analysis. In: 4th International Conference on Requirements engineering, 2000. Proceedings. pp. 7–16. IEEE (2000)
7. Merunka, V.: Instance-level modeling and simulation using lambda-calculus and object-oriented environments. In: Barjis, J., Eldabi, T., Gupta, A. (eds.) EOMAS 2011. LNBIP, vol. 88, pp. 145–158. Springer, Heidelberg (2011)
8. Merunka, V., Merunková, I.: Role of OBA approach in object-oriented process modelling and simulation. In: Barjis, J., Gupta, A., Meshkat, A. (eds.) EOMAS 2013. LNBIP, vol. 153, pp. 74–84. Springer, Heidelberg (2013)
9. Merunka, V., Nouza, O., Brožek, J.: Automated model transformations using the C.C language. In: Dietz, J.L.G., Albani, A., Barjis, J. (eds.) Advances in Enterprise Engineering I. Lecture Notes in Business Information Processing, vol. 10, pp. 137–151. Springer, Berlin (2008)
10. Miller, G.A.: The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol. Rev. **63**(2), 81–97 (1956)
11. Podloucký, M., Pergl, R.: The prefix machine – a formal foundation for the BORM OR diagrams validation and simulation. In: Barjis, J., Pergl, R. (eds.) EOMAS 2014. LNBIP, vol. 191, pp. 113–131. Springer, Heidelberg (2014)
12. Podloucký, M., Pergl, R.: Towards formal foundations for BORM ORD validation and simulation. In: Proceedings of the 17th International Conference on Enterprise Information Systems, pp. 315–322, April 2014
13. Silver, B.: BPMN method and style with BPMN implementer's guide: a structured approach for business process modeling and implementation using BPMN 2.0. Cody-Cassidy Press, Aptos (2011)
14. Yourdon, E.: Modern Structured Analysis. Yourdon Press, Englewood Cliffs (1989)

Chapter 16

Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment

[195] Náplava, P.; Pergl, R. Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment. In 2015 IEEE 17th Conference on Business Informatics, volume 2, July 2015, pp. 18–26.

Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment

Pavel Naplava
Faculty of Electrical Engineering
Czech Technical University in Prague
Technicka 2
160 00 Praha 6, Czech Republic
Email: naplava@fel.cvut.cz

Robert Pergl
Faculty of Information Technologies
Czech Technical University in Prague
Thakurova 9
160 00 Praha 6, Czech Republic
Email: robert.pergl@fit.cvut.cz

***Abstract*—Since 2009 approximately 500 business processes have been mapped and modelled at the Faculty of Electrical Engineering of the Czech Technical University in Prague. BPMN has been selected as the most suitable notation for the mapping purposes at the beginning of the project. The mapping has been done mostly by students from the Business Process Center of Excellence, a part of Faculty's Dean's office since 2009. This paper contains results of an experiment focused on possibilities of improving quality of existing results using the DEMO methodology. Next, we were interested in ways of enhancing students' skills of process modelling to create unambiguous and consistent models as a"result of a DEMO Bachelor course. Both students from the Center of Excellence and ordinary university students passed the DEMO Bachelor course of Enterprise Engineering Institute and their were given assignments of devising ontological analysis of the existing BPMN models. The results of the experiment are discussed in the paper, as well as considerations about future steps.**

Keywords: BPM, BPMN, DEMO, process mapping, process analysis

I. Introduction

All organizations try to find optimal ways to effectively realize their own business. It does not matter whether we are talking about a profit based or a nonprofit based subject. The problems are mostly the same – low efficiency, high costs, low response time, unsatisfied customers, etc. It can be simply said that any organization must solve two conflicting demands: improving output quality on the one side and lowering operating costs or raising the organization's income on the other. Academic institutions can primarily minimize operating costs only according to the legislative rules.

As it was presented in Hronza [1] and Náplava [2], in 2009 the Dean of the Faculty of Electrical Engineering at the Czech Technical University (CTU FEE) decided to optimize faculty operations (minimal costs, minimal redundancy of supporting activities, maximal automation of selected processes, full-cost model etc.) and started a transformation of the faculty becoming more of a business process oriented organization. The main reason was to prepare the faculty for the coming of new long term changes (reduced budget, lower number of students, wider industry cooperation, more scientific projects).

For the purposes of the faculty transformation project the dean established a new internal Business Process Center of Excellence (BPCE). Two fulltime employees work in the center, along with three PhD students and approx. 15 CTU students in the form of paid internships. All students have passed BPCE business process management courses and have been trained for the BPM skills. The student internships are between 2 and 3 years. New students are trained by the senior students.

The first BPCE's task was to analyze the existing successful transformation projects. The key results of the provided analysis are as follows:

- The supporting processes are the areas with the highest potential [3];
- It is neither possible nor necessary to transform all activities;
- It is necessary to start with processes that are not subjects of power interests [4];
- It is not reasonable to provide all changes at the same time;
- Transformation of any institution is a long term project.

The results of BPCE's investigation confirmed the assumptions regarding what must be done but did not give instructions on how to do it. This was the reason it was decided to create a new methodology based on the common BPM principles and practical experiences of the BPCE founders.

A. Business Process Management

Every transformation project starts with a deep understanding of the organization and its behavior. Different types of models are used for this purpose in practice. Through organization and behavior models it is possible to easily and visually identify potential problems and point out previously-unaware improvements needed to optimize the situation. Today, such models are called business processes and their management

is called Business Process Management (BPM). Some of the BPM's benefits are as follows [5]:

- Clear visibility and knowledge of an organization's activities;
- Definition of duties and roles within a company;
- Possibility to simulate and evaluate different business process scenarios;
- Identification of potential optimization areas;
- Ability to identify bottlenecks.

The current definition of BPM and business process have been dated to the early 1990s [5]. Hammer and Champy [6] defined a business process as "a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. A business process has a goal and is affected by events occurring in the external world or in other processes." This definition views a business process as a systematic ordering of work activities across time and place. Davenport [7] defined business process as "a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs: a structure for action." Both definitions omit the role of those performing work activities and their collaboration. These aspects were mentioned in Ould [8], who viewed a business process as:

- a set of purposeful activities;
- Activities are collaboratively carried out by a group of actors (human or machine);
- Activities can cross functional boundaries and are invariably driven by the outside world.

In this definition it is important to note that a business process is not carried out by a single individual or department only, but it can involve many different people or systems from different organizations. All are working together to achieve a common business goals. Business processes can be analyzed and optimized by either practical experiences or by scientific investigations. In order for the results obtained from the analysis to be reasonable it is necessary to have mechanisms more sophisticated than simple qualitative analysis of static diagrammatic models [9]. Aguilar-Saven [10] and Zakarian [11] recommend in their work the usage of formal techniques for an analysis of process models in order to make process modeling more attractive and meaningful. This formal approach to business processes modelling enables a measurement of the attainment of strategic goals and objectives by using performance indicators [12]. It is necessary always to think about both dynamic and functional aspects of the business process. According to van der Aalst [13] three different types of business process analyses exist:

- Validation; i.e., testing whether the business process behaves as expected in a given context;
- Verification; i.e., establishing the correctness of a business process;
- Performance analysis; i.e., evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization or other quantitative factors.

B. Business Process Modelling Notation

Many different notations for business process modelling in practice exist: BPMN, EPC, UML, IDEF0, etc. The main difference between them is in their usage; for example, UML is mostly used by software developers, BPMN is used by business analysts, etc. As the most suitable notation for the CTU FEE purposes BPMN (Business Process Modeling Notation) notation has been selected at the beginning of the transformation project.

The main reason why the BPMN was selected was the fact the notation is perceived as intuitive [14] and, based on the Semiotic Quality Framework applicaton, it can be easily learned for simple use and it is easy to understand [15].

II. Goals and Methodology

The goal of this paper is to provide a case study report of applying the DEMO method to improve business process modelling practice done in BPMN at the Faculty of Electrical Engineering of Czech Technical University in Prague. The current situation is that in recent 5 years, some 500 processes were analysed and mapped in BPMN. Because the follow-up goals of the CTU FEE's project are the mapping of the faculty's main processes, mapping of processes from other CTU's faculties and automation of the selected mapped processes all found issues must be solved. One of the possible solutions can be a combination of the BPMN and DEMO as it is described in [14]. At the same time, we wanted to leverage the opening of a new DEMO course for CTU's students. It seemed to be a way how to enhance the quality of the team and created results with minimal costs.

The structure of the paper is as follows:

We begin by a brief description of the current situation of BPM at the Faculty in section III. Next, we summarise the necessary theoretical background of the DEMO methology in section IV. We describe the setup of the performed study in section V and follow by its evaluation in section VI, which is the most important section. We mention some related work in section VII and finally, we draw conclusions and future plans in section VIII.

III. Initial Situation Description

Since 2009 BPCE has mapped and created approximately 500 process models (maps). In order for the project to be successful BPCE decided to start with a mapping of the dean's office processes and then present described process models to the whole faculty academia. It was planned for models to describe AS-IS state, but process owners provided a natural optimization too. This means owners did not strictly describe AS-IS state but they provided simple modifications based on their and other employees experience (ideas) and

created "pseudo" TO-BE models. After the first versions of created maps were accepted BPCE decided to add an extended form of the description. The reason was that the BPMN description is understood primarily by the owners and implementers of the process. Consumers of the process sometimes do not completely understand what to do and how to do it. Therefore new form of process description based on the BPMN and called "process as a service" has been added. The term "process as a service" can be simply described as a scenario of user actions. It represents a group of all related processes and information necessary for doing selected activity by employees, teachers or students. It combines both well known AS-IS and new, employees required, TO-BE activities. For example a simple scenario of education technical support is presented in fig. 1. Any teacher is able to see which technical support can he/she use during his/her lecture/seminar and how to ask for it.

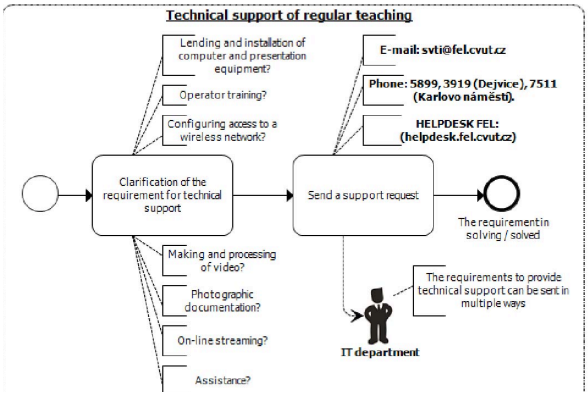


Fig. 1. Example of a process as a service.

Since 2009 approximately 150 processes as a service have been created. All created process maps and processes as a service are presented on the process portal that is a part of the faculty website. Anyone who is granted access to this portal can find any required information from his/her point of view, and can also provide feedback.

After 5 years of the CTU FEE's processes mapping BPCE has analyzed realized outputs and used method. The results are as follows:

- BPMN can capture business processes without the need of a special methodology.
- Created models are easily readable by business people.
- BPMN can be learned quickly and newbies can create new models after a short training of cca 20 hours in 10 weeks.
- Cooperation with students brings flexibility and creativity to processes' mapping.
- Only few elements of BPMN have been used for the models creation.

On the other hand:

- Selected created models can be used as an input for the process automation but the details of created models are insufficient.
- Nonexistence of the BPMN's models creation methodology and implicit model creation ontology results in ambiguous, inconsistent and incomplete models.
- Nonexistence of the unified ontology complicates validation of the models created by the different BPCE's members (mostly students), reusability of already created models for other academic subjects and uniform training of new BPCE members.

Because the follow-up goals of the CTU FEE's project are the mapping of the faculty's main processes, mapping of processes from other CTU's faculties and automation of the selected mapped processes, all the identified issues must be solved. One of the possible solutions can be a combination of the BPMN and DEMO as it is described in [14] and elaborated in the following section. Also the opening of the new DEMO course for CTU's students seemed to be a way how to enhance the quality of the team and created results with minimal costs.

IV. DEMO Method Overview

DEMO stands for "Design and Engineering Method for Organisations" and we consider it the leading methodology of Enterprise Engineering discipline, as it is grounded in scientific theories, mainly the system ontology of Bunge, teleology and the theory of communicative act of Habermas. At the same time, its benefits for the practical use has been proven, as documented e.g. in [16] or [17].

For our purpose, we selected DEMO because of the promises made in [18] (and acknowledged in later works). The paper's running example is similar to our situation in a respect that we have flowchart-based (BPMN) models, which we want to elevate to an enterprise engineering model and formulate conclusions. For his running example, Dietz concludes that:

1) *The ontological model of* (the running example) *is really and fully abstracted from the current way in which it operates... These properties, of course, make the ontological model very stable*, as organisational changes and specific ways of implementation will not change the model.
2) *The ontological model of* (the running example) *shows things that have no explicit implementation* – tacitly performed coordination acts are meant. We agree with Dietz that *these omissions are potential breakdowns in business processes as soon as changes are implemented*.

Moreover, Op't Land and Dietz in [16] showed that DEMO

- Offers a significant reduction of complexity (over 90% in terms of the size of documentation).
- It is an instrument for detecting tacitly performed coordination acts.

The above benefits seem very promising for us, as they address our concerns mentioned earlier. Thus we tried to apply DEMO for our case.

For a brief description of DEMO, we take a help of Op't Land and Dietz [16]:

A complete, so-called essential model of an organization consists of four aspect models: *Construction Model (CM)*, *Process Model (PM)*, *Action Model (AM)*, and *State Model (SM)*. The CM specifies the composition, the environment and the structure of the organization. It contains the identified *transaction types*, the associated *actor roles* as well as the information links between actor roles and transaction banks (the conceptual containers of the process history). The PM details each transaction type according to the *universal transaction pattern*. In addition, it shows the structure of the identified business processes, which are trees of transactions. The AM specifies the imperatively formulated *business rules* that serve as guidelines for the actors in dealing with their agenda. The SM specifies the *object classes*, the *fact types* and the declarative formulations of the *business rules*.

For our case, we focus on the two most fundamental models: CM and PM. Again, we let Op't Land and Dietz [16] explain:

A Construction Model shows the network of identified transaction types and the corresponding actor roles. E.g., transaction type `T01` delivers a business service to actor role `A00`. `A00` is called the *initiator* (consumer) and `A01` the *executor* (producer). The executor of a transaction is marked by a small black diamond on the edge of the actor role box. The solid line between `A00` and `T01` is the initiator link; the solid line between `A01` and `T01` is the executor link. fig. 2 also shows that some other actor role (`A07`) needs to have access to the history of transactions `T01` (production facts as well as coordination facts (e.g., status "requested", "promised", "stated", "accepted")). This is represented by the dashed line between `A07` and `T01`.

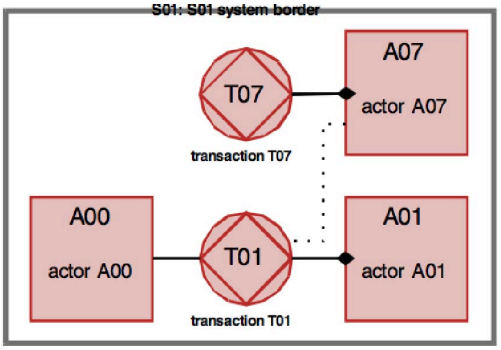


Fig. 2. Typical constructs of a DEMO Construction Model [16]

The DEMO Process Model reveals details of the transactions with the respect to universal transaction pattern. The so-called *standard transaction pattern* is depicted in fig. 3. The "happy flow" consists of `request`, `promise`, `state` and `accept`, as is depicted in fig. 4, which is also called the *basic transaction pattern*. Apart from the happy flow, `decline` may happen instead of `promise` and `reject` may happen instead of accept. Then, a new attempt may be made, or `quit`, resp. `stop` may end the transaction unsuccessfully. This logic is automatically included in all DEMO transactions, which is one of the reasons why the models are so compact – it would need a lot more diagram elements to express the transaction pattern of every transaction kind using a flowchart-like notation.

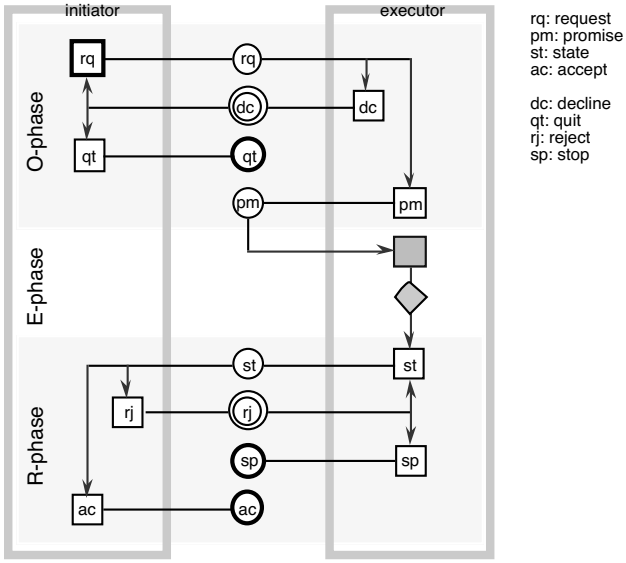


Fig. 3. DEMO Standard Transaction Pattern [19]

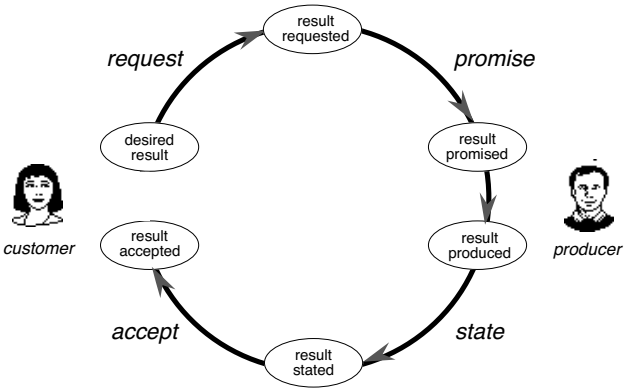


Fig. 4. Happy flow of a transaction [19]

Real situations may become even more complicated, which is addressed by the *complete transaction pattern* in fig. 5. It incorporates the notion of *revocation* – an actor may want to "take back" their act done before[1]. If that is allowed by the other party, the transaction "rolls back" to the desired state.

The complete transaction pattern is not depicted in CM nor PM, however we find interesting to incorporate it in our analysis.

[1]In the DEMO theory, nothing can disappear, so the original fact remains in the fact bank, however, the transaction flow is changed.

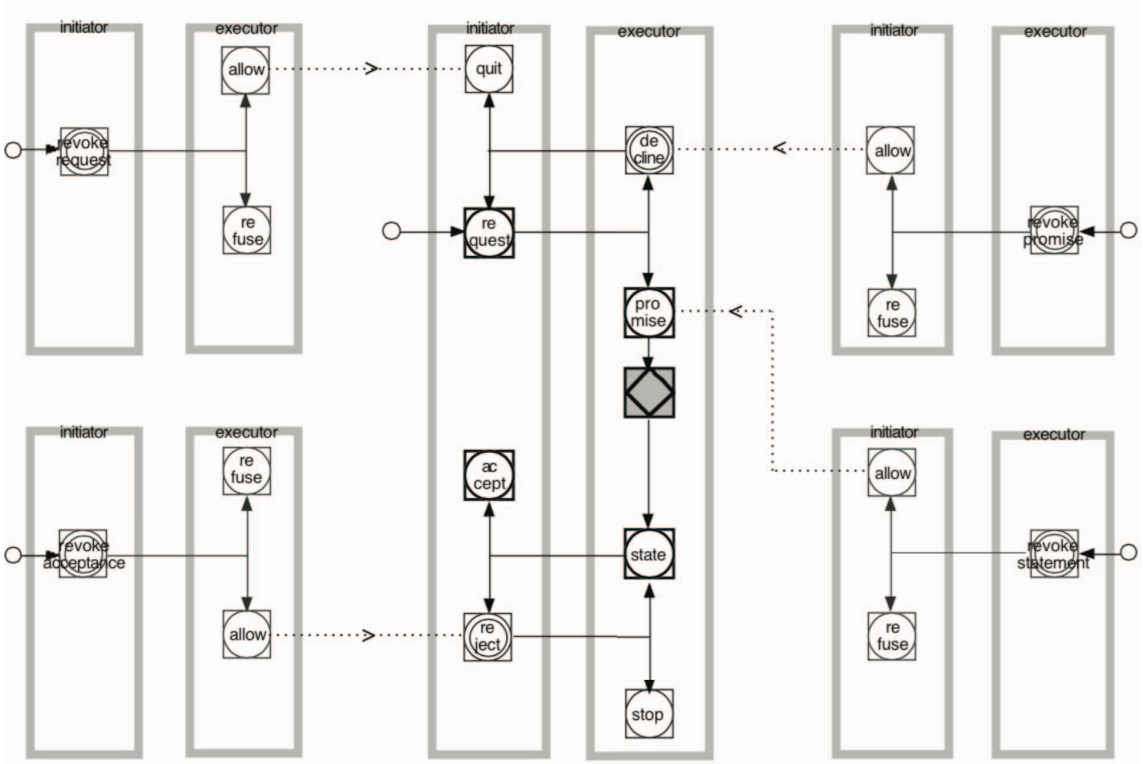


Fig. 5. DEMO Complete Transaction Pattern [20]

V. Study setup

The conditions under which the DEMO method was applied were:

- There is already a considerable amount of process modelling done, as described in section III.
- We have trained BPMN process analysts (both the BPCE and non-BPCE students) with no previous DEMO experience. They are master students.

The students were given a DEMO Bachelor training (we teach DEMO Bachelor course to our master students) prior to their work and then they were coached during their work by a DEMO Master.

There were 5 student teams of 5-6 students. Each team was assigned some area of processes. The tasks were:

1) Divide the existing process description into ontological, infological and datalogical parts.
2) Create Transaction Product Table (TPT) and Organisation Construction Diagram (OCD) at the ontological level.
3) Elaborate an analysis table for each transaction.
4) Create a sample of PSD (Process Structure Diagrams, [21] for about 7 transactions.
5) Create a necessary OFD.
6) Show two interesting AM descriptions.
7) Create an actor - actor role mapping table.
8) Formulate the conclusions.

The student teams analysed totally 34 processes spanning in size from a few activities to an A4 scheme.

VI. Study Evaluation

A. Task 1

The first task, discerning the existing process description into ontological, infological and datalogical parts was carried out by "colouring" the BPMN diagrams using red, green and blue colours, similarly to the running example in [18]. A sample colouring is shown in fig. 6.

The overall results were:

- 124 ontological parts of the models
- 79 infological parts of the models
- 76 datalogical parts of the models

By "parts" here we mean BPMN elements, mostly tasks and gateways. The students had generally no difficulties in proper distinguishing the three levels. The results show that in spite of a lack of solid engineering method during the process modelling, the analysts' training moved them into "preferring" the ontological matter in their diagrams. However, using these rough numbers, we see that the ontological essence is still just some 44% of the diagrams.

B. Task 2

Overall, 58 ontological transactions were identified by the teams and verified by the DEMO Master. Here, the students

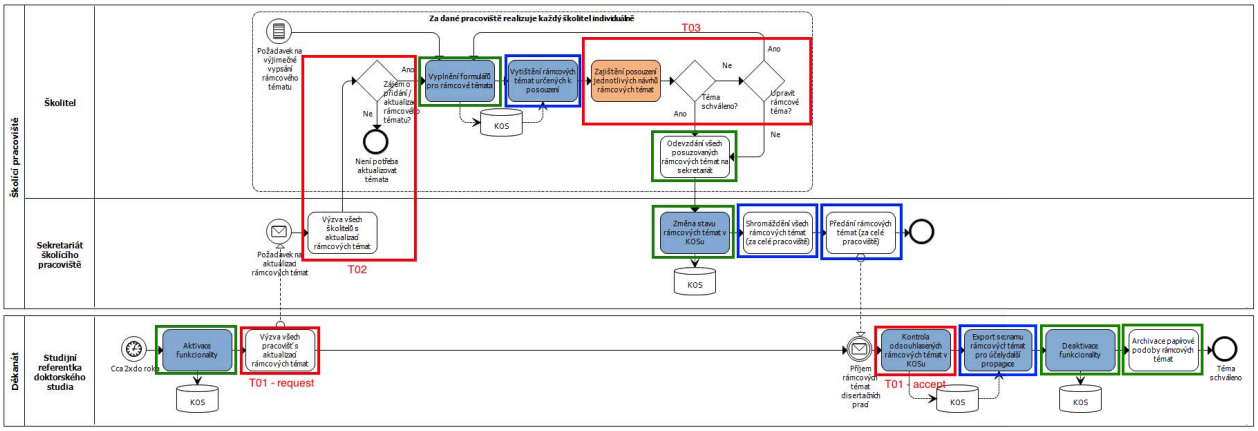


Fig. 6. Sample BPMN process colouring

Transaction ID:	T03
Transaction Name:	Admission Fee Payment
Product:	The admission fee has been paid
Initiator:	The admission clerk
Request:	*unclear*
Promise:	*unclear*
State:	Stating the payment document during the admission process
Accept:	Accepting the payment document
Decline:	*not modelled*
Reject:	*not modelled*
Revoke request:	*not modelled*
Revoke promise:	*not modelled*
Revoke statement:	*not modelled*
Revoke acceptance:	*not modelled*

TABLE I
TRANSACTION ANALYSIS TABLE

needed some practice in formulating properly the transaction kinds and especially the products. It came out that although a great deal of attention was paid to these constructs during the lectures, several rounds of practice and feedback were necessary for this skill to evolve sufficiently.

Now, if we compare the number of parts of the orignal models, which were analysed, i.e. $127+79+76 = 279$ with the resulting number of 58 transactions, we come to a result that the analysed models' essence is 21% of the original amount of model parts. Again, it is above the officially stated 10%, showing that the BPMN analysts did an above-average job in modelling the essence.

C. Task 3

table I shows an example of a transaction analysis table, which were elaborated for each transaction kind based on the original BPMN models.

We can say that transaction analysis tables were the most interesting part of the outputs. They clearly show how various parts of the complete transaction pattern were covered by the existing models. Here, we confirm that mostly happy paths were described, with occasional unhappy branches in the most typical situations. This was an expectable result, however, we were quite surprised, that sometimes even parts of the happy path were missing, like in table I, which describes the admission fee payment transaction that happens as a part of student enrolment into the study programme admission process. In the BPMN diagram, it was supposed that the student states the payment document during the admission process, however the corresponding request was nowhere to be found. Discussions with stakeholders were not part of the project, so we do not know for sure, but we expect that it is supposed that the student just "knows he has to pay" from the instructions on the web.

We find the transaction analysis table being a powerful device for models improving, as they clearly show transaction parts that are described unclear or not at all. Of course, for many situations, it makes sense not to allow the revoke acts, however it is just beneficial to ponder these situations during the modelling.

D. Tasks 4, 5, 6

The results of these tasks were standard DEMO PSD, OFD and AM models. Students clearly struggled with OFD and especially the AM models, so we will not discuss them here. PSD models often revealed additional discussions about the precise binding of the transaction steps – "is statement enough, or do we need the acceptance of Tx to promise Ty"? These questions are clearly a positive effect of the DEMO training and raising them leads to more accurate process models.

E. Task 7

The actor - actor role mapping table is shown in table II, where "A" means that the actor is responsible for a certain

actor role, while "D" means that the actor is delegated to perform a certain actor role.

In spite of being not so much impressive achievement at the first sight, we think that this table actually may have the strongest impact at the managerial level. The situation is that very often during the process mapping, responsibility and authority discussions take place, which complicates the modelling, causes confusions and delays. It has even happened at one faculty that this type of discussion killed the efforts of process management entirely. DEMO distinguishes very strictly between the actor roles, being the holders of responsibility and authority and actual actors. Using this separation, the analysis and political discussions are separated – they take place just over the mapping table in the second step.

At the same time, this separation of actor roles from actors caused really hard time to our students. In the end, they were able to make the paradigm shift, however it was definitely one the most hard ones for them. They were fluent in telling apart "Alena" from "secretary", however this additional abstraction level needed a considerable time and practice to sink.

F. Task 8

From the variety of conclusions the students made, let us pinpoint the ones we find the most interesting and beneficial to take into account:

- All teams complained that often they were not able to tell apart ontological, infological and datalogical parts of the diagrams. It showed that this was not due to lack of their knowledge and training, but due to lack of information in the models, which would need some additional clarifications from the stakeholders, e.g. "preparing the list of students" could have been primarily ontological, infological or a datalogical act depending on the competences exercised.
- a lot of confusion was about the transaction analysis table. The students filled in the obvious information expressed in the diagrams, however then a lot of questions remained. Additional discussions with the stakeholders would be necessary to clarify them.
- Sometimes it was possible to map the "elements flow" in the original BPMN diagrams to the parts of the transaction pattern in a mostly straightforward way, however sometimes it was quite hard to put the "puzzle" together and identify certain BPMN tasks as transaction pattern coordination acts.
- Some hard times were posed to students by the fact that some of the original BPMN models were modelled from the perspective of one instance of a student, like making a decision about accepting them into a study programme, while the others were the "mass" ones, like carrying out the admission tests. It showed that BPMN is quite a comfortable platform to mix and interweave these levels, DEMO does not tolerate vagueness in this form.

VII. Related Work

Nuffel, Mulder and Kervel provide a thorough study of enhancing the formal foundations of BPMN by Enterprise Ontology in [14]. Other efforts to enhance existing modelling methodologies by DEMO were done for ArchiMate [22] and ARIS [23].

An important aspect of methodology change is also its adoption. Kris and Verelst in [24] discuss various challenges and tasks that are related to adoption of the DEMO methodology by organisations (a research agenda).

At the practical level, we may mention efforts of Czech Ministry of Education. To assist Czech academic and research institutions, the Ministry carried out a project called "Efficient Institutions" (EFIN) during the years 2009–2012 [25]. Its goal was to help institutions analyse their present situation and propose which management methods be used in the key institution's areas to optimize their operations. During the project's discussions and workshops it showed that some institutions had attempted to start transformation activities, but all of them failed sooner or later. The main reason was that no methodology on how to successfully put a transformation project into practice existed. This fact was confirmed by representatives of 8 from the total 26 Czech public higher education institutions participating in the EFIN project [26]. As for the private institutions, only 2 of the total 45 [27] were participating in the project but they also confirmed that attempts to transform the institution were unsuccessful. Also, the analysis of foreign academic institutions did not find any generally usable methodology [28]. It has been only found that the standard managerial methods usually used in commercial organizations are commonly used in these institutions too. The most important finding was that their operations were based on the process management (BPM) principles.

VIII. Conclusions and Future Work

The study setup was quite different from the most DEMO-application studies in the sense that the analysts were not seasoned professionals working at big enterprise projects, but they were master students just after getting the DEMO Bachelor course. We can conclude that even with this "lightweight" training process, the results showed positive effects of the training. Specifically, the students exhibited

- Better skills in precise formulations (better naming and expressions).
- The ability to discern ontological, infological and datalogical levels and their importance.
- It is interesting that the original analysts performed an above-average job while doing the BPMN modelling, as the essential DEMO model is usually 10% of the original model. In our case, it is 21% of the original model.

On the other hand, the students of course lacked the experience of DEMO professionals, which showed especially in:

- They are not able to reach the most essential and simple models.

	Referentka SO	Žadatel o studium	Proděkan pro BS	Vedoucí KM	Zaměstnanec KM	VIC	Vedoucí SO	Organizátor ZK
A01							A	
A02		A						
A03	D						A	
A04							A	
A05		A						
A06				A				
A07				A	D			
A08	D						A	
A09						A		
A10								A
A11		A						
A12				A				
A13				A	D			
A14							A	
A15			A					
A16		A						
A17	D						A	
A18		A						
A19			A					

TABLE II
THE MAPPING BETWEEN THE TRANSACTION ACTOR ROLES (ROWS) AND SPECIFIC FACULTY POSITIONS (COLUMNS)

- They struggle with proper formulations of transaction products.
- They struggle with separating actor roles and actors.

It would be interesting to draw some conclusions about the correlation between the level of DEMO adoption versus the previous process modelling experience and skills. Howerver, we do not dare to draw such conclusions, because there are a lot of aspects that may affect the correlation significantly, as

- Our BPMN-trained students tend to be "above-average" students with a high motivation to learn new things.
- Some of them are working on diploma theses related to process modelling.
- They are analytical types, while some of the others were rather programmers.

Our feeling is that providing more theoretical lectures to the students would not raise the quality of their results, on the other hand, providing more exercises and practical training has this potential. It would be interesting to do some further research on this.

At the general level, the study helped us considerably with formulating current shortcomings of our analysis:

- There are confusions about distinguishing red, green and blue levels – for some activities, like report generation, it is not generally clarified to which level does it belong.
- Negotiating about competences – right now, process mapping is sometimes blocked by petty discussions who will do what.
- Improving BPMN models using DEMO analysis requires participation of the stakeholders: e.g. sometimes it is necessary to clarify some concepts to decide about the ontological level.

The following research steps will start with the detailed analysis by

- studying the feedback from the last run of the DEMO course,
- results of master and bachelor thesis dealing with:
  - BPMN and DEMO comparison and application from the student's point of view,
  - BPMN and S-BPM comparison and application from the student's point of view,
  - BPMN and UML comparison and application from the student's point of view.

The goal of the analyses is to identify general BPM abilities of technically oriented students (CTU's students) and evaluate differences between BPM experienced and non-experienced students. Based on the analysis results, the main interests (hypothesis) of the future research are as follows:

- Is it possible to minimize student's lack of practical DEMO experience? Is there a way how to extend basic DEMO course, for example by adding one or more practical seminars, to minimize student's confusions in practical DEMO methodology application? These questions are a part of a big question, how to properly educate students with of entry levels of BPM knowledge and skills.
- How to leverage the potential in the DEMO methodology

for our current situation of process mapping at our University? Mostly, how to extend the present mapping methodology to improve it.

- The next generation of the methodology will probably need to specify various analysts roles according to their skills and tasks.
- How to balance the resulting models to be complete and detailed to become valuable input for enterpise and software engineering processes, while at the same time make them easily understandable by common users, to make them a knowlege base and a communication medium, as well?
- Is it possible to elaborate a set of selected faculty processes that can be generally used in any faculty both inside and outside the CTU? Is the DEMO methodology an appropriate tool for increasing ratio of reusable processes? What are the main problems of academic processes reusability? Can DEMO help to identify process reusability bottlenecks?

REFERENCES

[1] R. Hronza and M. Speta, "Business Process Center of Excellence at the Faculty of Electrical Engineering at the Czech Technical University in Prague," in *2013 IEEE 15th Conference on Business Informatics (CBI)*, Jul. 2013, pp. 346–349.

[2] P. Náplava, R. Hronza, J. Kočí, and J. Pavlíček, "How to Successfully Start the Transformation of an Academic Institution Case study on the process mapping project at the Czech Technical University," in *Complementary proceedings of the 8th Workshop on Transformation & Engineering of Enterprises (TEE 2014), and the 1st International Workshop on Capability-oriented Business Informatics (CoBI 2014) co-located with the 16th IEEE International Conference on Business Informatics (CBI 2014)*, vol. 1. Geneva: Aachen: RWTH Aachen University, 2014, pp. 1–15.

[3] K. Schedler and U. Helmuth, "Process management in public sector organizations," in *Public Management and Governance*. Routledge (Oxon (UK)), 2009, vol. 2, pp. 181–198.

[4] B. Adenso-Díaz and A. F. Canteli, "Business Process Reengineering and University Organisation: a normative approach from the Spanish case," *Journal of Higher Education Policy & Management*, vol. 23, no. 1, pp. 63–73, May 2001.

[5] R. K. L. Ko, "A Computer Scientist's Introductory Guide to Business Process Management (BPM)," *Crossroads*, vol. 15, no. 4, pp. 11–18, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1558897.1558901

[6] M. Hammer and J. Champy, "Reengineering the corporation: A manifesto for business revolution," *Business Horizons*, vol. 36, no. 5, pp. 90–91, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0007681305800643

[7] T. H. Davenport, *Process Innovation: Reengineering Work Through Information Technology*. Boston, MA, USA: Harvard Business School Press, 1993.

[8] M. A. Ould, *Business Processes: Modelling and Analysis for Re-Engineering and Improvement*. New York, NY: John Wiley & Sons, May 1995.

[9] K. Vergidis, A. Tiwari, and B. Majeed, "Business Process Analysis and Optimization: Beyond Reengineering," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 1, pp. 69–82, 2008.

[10] R. S. Aguilar-Savén, "Business process modelling: Review and framework," *International Journal of Production Economics*, vol. 90, no. 2, pp. 129–149, Jul. 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527303001026

[11] A. Zakarian, "Analysis of Process Models: A Fuzzy Logic Approach," *The International Journal of Advanced Manufacturing Technology*, vol. 17, no. 6, pp. 444–452, Apr. 2001. [Online]. Available: http://link.springer.com/10.1007/s001700170162

[12] C. Lewis, "A source of competitive advantage?" *Management Accounting (GB)*, vol. 71, no. 1, pp. 44–46, 1993.

[13] W. M. P. van der Aalst, "Business Process Management: A Personal View," *Business Process Management Journal*, vol. 10, no. 2, pp. 135–139, 2004.

[14] D. Van Nuffel, H. Mulder, and S. Van Kervel, "Enhancing the Formal Foundations of BPMN by Enterprise Ontology," in *Advances in Enterprise Engineering III*, W. van der Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, A. Albani, J. Barjis, and J. L. G. Dietz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 34, pp. 115–129.

[15] O. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *IEEE Software*, vol. 11, no. 2, pp. 42–49, Mar. 1994.

[16] M. Op 't Land and J. L. G. Dietz, "Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations," in *Advances in Enterprise Engineering VI*, W. van der Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, C. Szyperski, A. Albani, D. Aveiro, and J. Barjis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 110, pp. 77–92.

[17] 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 *Advances in Enterprise Engineering VIII*, W. van der Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, C. Szyperski, D. Aveiro, J. Tribolet, and D. Gouveia, Eds. Cham: Springer International Publishing, 2014, vol. 174, pp. 16–30.

[18] J. L. G. Dietz, "ENTERPRISE ONTOLOGY - UNDERSTANDING THE ESSENCE OF ORGANIZATIONAL OPERATION," in *Enterprise Information Systems VII*, C.-S. Chen, J. Filipe, I. Seruca, and J. Cordeiro, Eds. Dordrecht: Springer Netherlands, 2006, pp. 19–30.

[19] ——, *Enterprise ontology: theory and methodology*. Berlin; New York: Springer, 2006.

[20] Jan L.G. Dietz, *THE ESSENCE OF ORGANISATION - AN INTRODUCTION TO ENTERPRISE ENGINEERING*. Sapio bv, 2012, 00000.

[21] ——, "DEMO-3 Models and Representations (version 3.7)," Jan. 2014.

[22] R. Ettema and J. L. G. Dietz, "ArchiMate and DEMO – Mates to Date?" in *Advances in Enterprise Engineering III*, W. van der Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, A. Albani, J. Barjis, and J. L. G. Dietz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 34, pp. 172–186.

[23] D. Strijdhaftig, "On The Coupling Of Architectures: Leveraging DEMO Theory Within The ARIS Framework," Ph.D. dissertation, TU Delft, Sep. 2008.

[24] K. Ven and J. Verelst, "The Adoption of DEMO: A Research Agenda," in *Advances in Enterprise Engineering III*, W. van der Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, A. Albani, J. Barjis, and J. L. G. Dietz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 34, pp. 157–171.

[25] MinistryOfEducationYouthAndSports, "Projekt EFIN, Analyza 2: Zkusenosti a vhodne pristupy efektivniho (in Czech)," Jan. 2015. [Online]. Available: http://efin.reformy-msmt.cz/download-document/efin-analyza-2-cs-project

[26] ——, "Public Higher Education Institutions – Websites," Jan. 2015. [Online]. Available: http://www.msmt.cz/areas-of-work/tertiary-education/public-higher-education-institutions-websites

[27] ——, "Overview of Private Higher Education Institutions," Jan. 2015. [Online]. Available: http://www.msmt.cz/areas-of-work/tertiary-education/overview-of-private-higher-education-institutions

[28] O. Kubera, "Analysis and Optimization of Faculty Processes (in Czech)," Master Thesis, CTU FEE Prague, 2012.

Chapter 17

Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method

[196] Mráz, O.; Náplava, P.; Pergl, R.; et al. Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method. In Advances in Enterprise Engineering XI: 7th Enterprise Engineering Working Conference, EEWC 2017, Antwerp, Belgium, May 8-12, 2017, Proceedings, Cham: Springer International Publishing, 2017, ISBN 978-3-319-57955-9, pp. 85–98.

Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method

Ondřej Mráz, Pavel Náplava, Robert Pergl, Marek Skotnica

Czech Technical University in Prague, Czech Republic
mraz.ondra@gmail.com, naplava@fel.cvut.cz
perglr@fit.cvut.cz, skotnicam@fit.cvut.cz

Abstract. The goal of this paper is to contribute to efforts of improving the Business Process Modelling (BPM) practice. We present an original method for converting enterprise ontology Design & Engineering Method for Organisations (DEMO) process models into a BPMN 2.0 notation. By this approach, we are able to mitigate certain methodological deficiencies of BPMN. The method exhibits the following qualities: Implementation of the complete transaction pattern formulated by the PSI-theory, correct managing of multiple child transaction instances, and executability of the resulting BPMN model.

Key words: PSI-theory, BPMN, DEMO, Business Process Modelling, Enterprise Ontology, Conceptual Modelling

1 Introduction

BPMN (Business Process Model and Notation) [1] is a graphical notation that is used for modelling business processes. Key characteristics of BPMN are simplicity of the underlying theory (flowchart), standardised notation and a large number of tools. This makes BPMN one of the most wide-spread process modelling notation in practice, in spite of its limitations and flaws. BPMN offers three different types of diagrams: Choreography, Conversation and Collaboration diagrams. For this work, only the Collaboration Diagram will be considered. This diagram expresses the process flow in achieving participants' goals.

One of the BPMN weaknesses is the absence of a methodology for constructing diagrams, which is addressed for example by Silver [2]. Nevertheless, the design freedom is still too broad, which results in different modelling styles of individual analysts and different models depicting the same situation, which complicates enterprise engineering tasks like mergers and reorganisations.

DEMO (Design & Engineering Method for Organisations) [3] is a leading modelling method used in the discipline of Enterprise Engineering [4] based on deep and sound theoretical basis (the PSI-theory) and high ontological relevance. Its benefits for the practical use has been proven, as documented for example in [5] or [6]. It does not limit itself just to process modelling, but it also deals with capturing structural (factual) knowledge and business rules, thus delivering a complete enterprise ontology exhibiting certain criteria (C4E). However,

compared to BPMN, DEMO is still a niche approach and relatively demanding to master. Also, a limited number of tools is available today.

For a brief description of DEMO, we take a help of Op't Land and Dietz [5]:

A complete, so-called essential model of an organization consists of four aspect models: *Construction Model (CM)*, *Process Model (PM)*, *Action Model (AM)*, and *State Model (SM)*. The CM specifies the composition, the environment and the structure of the organization. It contains the identified *transaction types*, the associated *actor roles* as well as the information links between actor roles and transaction banks (the conceptual containers of the process history). The PM details each transaction type according to the *universal transaction pattern*. In addition, it shows the structure of the identified business processes, which are trees of transactions. The AM specifies the imperatively formulated *business rules* that serve as guidelines for the actors in dealing with their agenda. The SM specifies the *object classes*, the *fact types* and the declarative formulations of the *business rules*.

The DEMO Process Model reveals details of the transactions with the respect to universal transaction pattern. The basis is the "happy flow" consisting of `request`, `promise`, `state` and `accept`, as is depicted in fig. 1, which is also called the *basic transaction pattern*. In the so-called *standard transaction pattern* (not depicted), `decline` may happen instead of `promise` and `reject` may happen instead of accept. Then, a new attempt may be made, or `quit`, resp. `stop` may end the transaction unsuccessfully.

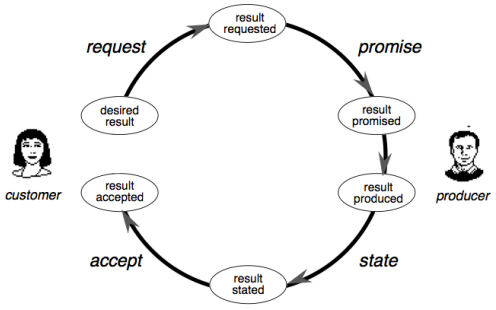


Fig. 1. Happy flow of a transaction [3]

Real situations may become even more complicated, which is addressed by the *complete transaction pattern* in fig. 2. It incorporates the notion of *revocation* – an actor may want to "take back" their act done before[1]. If that is allowed by the other party, the transaction "rolls back" to the desired state.

The logic of the complete transaction pattern is automatically included in all DEMO transactions, which is one of the reasons why the models are compact.

[1] In the DEMO theory, nothing can disappear, so the original fact remains in the fact bank, however, the transaction flow is changed.

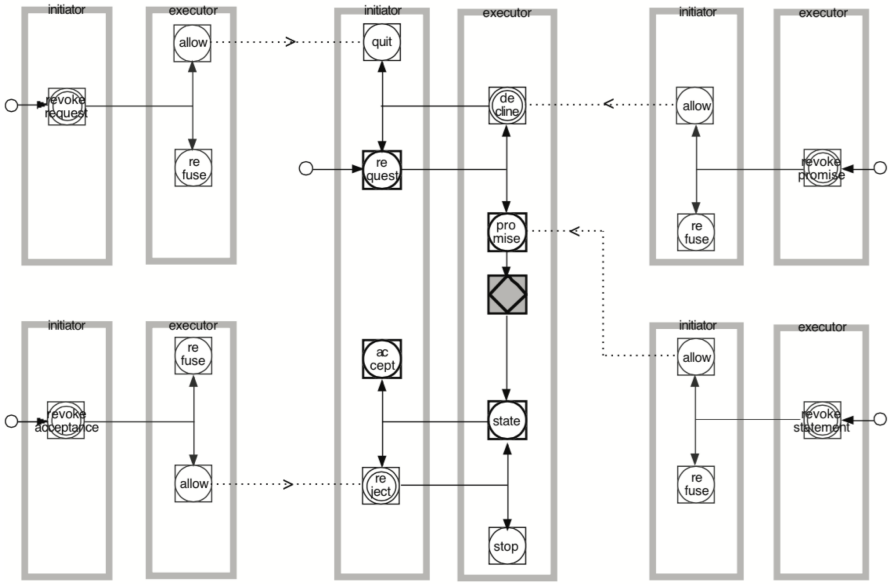


Fig. 2. DEMO Complete Transaction Pattern [7]

The main goal of this paper is to combine the simplicity of the BPMN and ontological qualities of the DEMO. The result is the method that converts enterprise ontology Design & Engineering Method for Organisations (DEMO) process models into a BPMN 2.0 notation. This approach mitigates the mentioned absence of a sound methodological approach for BPMN. The BPMN models resulting from the described method converge, similarly to DEMO, to one essential model, thus eliminating different modelling styles of individual analysts leading to comparable models. Our other requirements are: implementation of the complete transaction pattern formulated by the PSI-theory, correct managing of multiple child transaction instances, and executability of the resulting BPMN model.

We start the paper by the discussion of the related work of efforts of improving BPM and BPMN, specifically the approaches based on applying the enterprise-engineering rigour (section 2). We then briefly present results of a comparative analysis of DEMO and BPMN (section 3), which led to formulating our method of conversion (section 4). We demonstrate the method on an example (section 5). Finally, we discuss the result and formulate conclusions (section 6).

2 Related Work - Improving BPM and BPMN

A poor ontological quality of BPMN is generally known and documented [8]. The most practised remedy is exercising a methodological approach like the one proposed by Silver [2], who distinguishes three levels of BPMN: (i) Descriptive,

(ii) Analytical and (iii) Executable and proposes several analysis patterns and anti-patterns.

The discipline of enterprise engineering (EE) [4] brought about a rigorous approach of building an enterprise ontology (EO) [3], DEMO being its modelling method. There are several foundational EE theories, the most notable being the PSI-theory. As on of the central concerns of EE is the business process management, the effort to apply EE theories (EET) to the existing (less formal approaches) is promising. The efforts in this area are twofold:

1. Applying EET for analysis of existing BPMN models of business processes: for example [9], [10] and [11].
2. Enhancing the formal foundations of BPMN by EET: for example [9], [10], [12], [13], [14]

2.1 Applying EET for Analysis of Existing BPMN Models of Business Processes

Caetano et al. showed that applying the DEMO PSI-theory to improve business process modelling deserves attention [9]. The authors started by analysing existing BPMN models and identified missing DEMO transaction pattern steps in these models. It had been determined or each BPMN activity from the analysed models, whether this activity is an ontological, infological or datalogical part of a transaction. It had been also determined, which part of the transaction pattern each activity represents. Next, the authors created an ATD and a PSD diagram of DEMO and using a PSD diagram, they enriched existing BPMN models by adding missing parts of the transaction pattern into the BPMN models.

In the second part, the authors present results of applying this method to analysis of existing BPMN models of key processes of a big organization (more than 500 activities and 60 actors). The authors identified numerous missing act types in the original BPMN models. The results from this analysis were:

– 25% of production C-acts missing in the original BPMN model,
– 25% of request C-acts missing in the original BPMN model,
– 50% of promise C-acts missing in the original BPMN model,
– 25% of state C-acts missing in the original BPMN model,
– 40% of accept C-acts missing in the original BPMN model.

Results reported by Pergl and Náplava for an academic institution [11] state reduction of DEMO essential models complexity to 21% of the original BPMN size and several model quality improvements similar to [9].

2.2 Enhancing the Formal Foundations of BPMN by EET

These efforts aim to precisely express the EE ontological constructs using the standard BPMN notation. Two approaches have been followed. The first is to enhance the BPMN models by adding the missing C-(F)acts and other constructs from the PSI-theory. Caetano [9] is an example of this method.

The second way is generating BPMN models from the DEMO models. This method was discussed in two diploma theses [13], [14], from which the approach in this paper was designed.

3 Analysis of DEMO and BPMN

Here follows observations of comparing various aspects of DEMO with respect to BPMN, from which follows the conversion principles and decisions made. These were formulated based on the DEMO theory axioms and models definitions related to the BPMN elements definitions, as introduced in section 1.

- Similar parts of methods that can be simply transformed from the DEMO to BPMN:
  - The *Process Structure Diagram (PSD)* of DEMO contains process information, which can be related to a BPMN process diagram.
  - The *Action Model (AM)* of DEMO expresses complex decision rules for Coordination acts (C-acts)[2]. The contained information can be used for branching in BPMN.
  - BPMN does not distinguish the three key human abilities (forma, informa, performa), however applying this distinction can be introduced straightforwardly, as shown for example in [11]. As this concern is orthogonal to our effort, we do not discuss the distinction axiom here.
  - Related to the point above, the (atomic) actor roles in DEMO are executors of exactly one transaction, while swimlanes may contain many different actions.
- Different parts of methods that require deep analysis before transformation from DEMO to BPMN:
  - The DEMO *Transaction Axiom* concept does not exist in BPMN. Only happy flows and the most obvious unhappy flows are expressed in models.
  - The *Object Fact Diagram (OFD)* being a factual model does not have an analogy in BPMN.
  - DEMO and BPMN employ different execution models. While BPMN is flow-based, DEMO operates on the basis of a so-called CRISP model [3], which may be characterised as an event-driven, or more precisely, an agenda-driven execution model.
  - The *Construction Model (CM)* of DEMO is an abstraction that does not specify process, it provides just structural information.

4 Converting DEMO into BPMN

The goal is to convert the complete transaction axiom into BPMN, including all revoke types. Sections 4.1 to 4.4 describe all the necessary pieces and section 4.7 presents the result. We used BPMN 2.0 and leveraged the newly available *Data Store* construct.

[2] Apart from containing all the information from the other models.

4.1 C-acts

C-acts are essentially activities that take place in order specified by the transaction pattern. BPMN has the concept of *activities* and the order is specified by *sequence flows*. As C-acts are atomic, the appropriate activity type is *task*.

4.2 C-facts

As mentioned in section 1, a C-fact becomes existent in the world as a consequence of performing a C-act. Heller in his thesis [13] lists three possibilities of expressing C-facts using BPMN:

1. Not explicitly expressed – the existence of the fact-C is not explicitly expressed. It is indirectly realised by a sequence flow. This option is sufficient if revokes are not considered (see further).
2. Using a BPMN message – the actor, who performs the given C-act sends a BPMN message with the C-fact to the other actor (transaction participant).
3. Using a BPMN signal – the actor, who performs the given C-act emits a BPMN signal on creating a C-fact. This has the benefit that apart from the other actor, any other actor may subscribe to the signal reception, which is aligned with the PSI-theory, where facts are present in the world, not only in the transaction, thus available also outside the transaction (modelled by interstriction links).

However, under a closer consideration, none of the above solutions are completely sufficient for a correct handling of revokes. For each revoke act, the PSI-theory specifies a certain *state* in which the transaction must be. The state is formulated like "X or further": request(ed) or further, promise(d) or further and so on. This is why we decided for another representation: the BPMN 2.0 *data store*, into which the state of the transaction is stored. This data store is connected to every C-act activity by an association.

4.3 P-(F)acts

It is not necessary to store information about them creating a P-(f)act into the data store, because they can be derived from C-(f)acts: According to the PSI-theory, the P-fact starts to exist based on acceptance of the product, so P-(f)acts can be expressed by an activity only. If need be (optimisation of an implementation), they can be stored similarly to the C-(f)acts described above.

4.4 Actors

Swimlanes in BPMN are isomorphic to actors in DEMO [10]. BPMN lacks a higher abstraction level of actor roles, being the logical sum of responsibility, authority and competence necessary to carry out the product [3]. There are generally two approaches: (i) abstracting the swimlanes to actor roles (like Decider

or Concluder), (ii) remaining on the BPMN's low level of abstraction and using swimlanes to represent actors – company functional roles – like CEO or specific people like Jane.

Another possibility for representing actor roles is using BPMN *pools*, where each pool represents one actor. The resulting BPMN models will be very similar to models using swimlanes, however we have not chosen this representation because: (i) The correspondence of actor roles and transactions is not explicit, (ii) sequence flow cannot be used between pools, which would result in using messages, further complicating the diagrams.

4.5 The Composition Axiom

A composition of transactions may be dealt with in two ways: (i) to model all the transactions in one diagram, (ii) to separate diagram for every transaction. Generally, both approaches are valid, but (ii) may lead to huge diagrams, as can be seen in fig. 10 and fig. 11. As (ii) guarantees the limit of the diagram size, we preferred it. On the other hand, it may render understanding of the big picture harder.

We propose the following 2-part expression of the composite axiom:

1. **Launching a child transaction** in a specific place in the parent transaction. The child transaction must be started just after creating a specific C-fact. A *message-throwing event* may be used in case of initiating a single child transaction. In case of firing multiple child transactions, signals are appropriate, similar to the C-acts above. Moreover, it is needed to ensure the multiplicity. In case that it is greater than one, we need to initiate several child transaction instances. This is achieved either by using a *cycle* for creating child transactions or a *loop activity*. Modelling by cycle (fig. 3) means, that the model contains an activity counting, how many times the activity was run. After this activity, there is a gateway. If the counter has not reached the number of child transaction instances to spawn, the process goes into message throwing event to start a child transaction instance and then the process returns to the counting activity. This happens $0..N$ times, as required. When multiplicity is modelled by a loop activity (fig. 4), the activity is in the form of a subprocess (with parallel loop), which sends a signal[3] that starts a child transaction. In the examples described below, the first (counter) variant is used because the model is more explicit. At the same time for models with a multitude of child transactions, the more concise loop variant is recommended. Also, from the execution point, the implementation variant may be driven by the vendor, as correlation of instances must be ensured (more discussed in section 4.8).
2. **Blocking execution of the parent process** until the child process has not reached the given state (creating a C-fact being waited on). This blocking can

[3] We cannot use a message send in this situation, because the encapsulation would be violated.

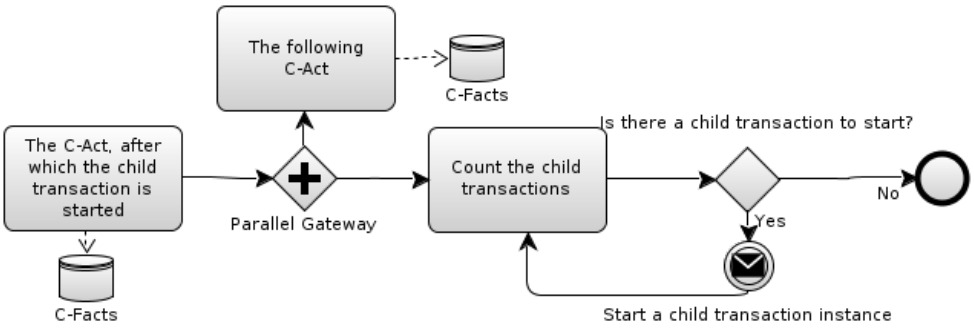


Fig. 3. Launching child transactions by using counter

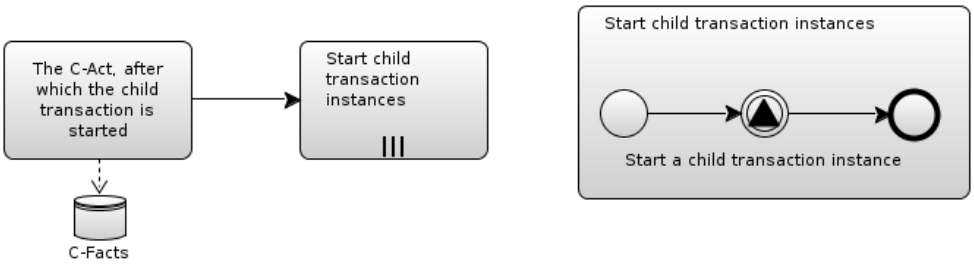


Fig. 4. Launching child transactions by using loop

be realized by a BPMN *catching event condition* in the parent process waiting for a specific condition before the given C-act. Here, a conditional event must be used instead of a signal event, as we do not wait just for a signal, but for a specific instance in case of multiple child transaction instances. This situation is modelled in fig. 5. Again, specific vendor correlation techniques may apply (section 4.8).

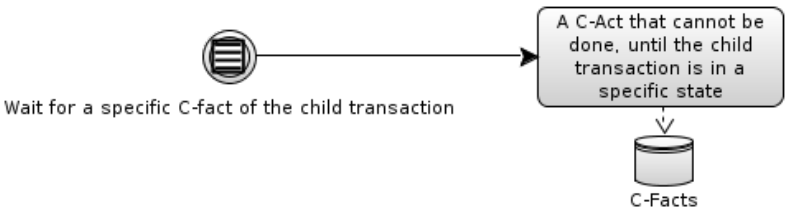


Fig. 5. Waiting for a child transaction

4.6 Revokes

Revokes are the most challenging part of the conversion. Let us present the challenges and how we dealt with them:

– A revoke must be applied on a specific instance of the transaction; in a certain time, there can be several parallel transaction instances running. This must be ensured by the BPMN system (section 4.8).
– A revoke can be fired independently on the running main process. Can be modelled straightforward, as BPMN allows several independent start events.
– A revoke can be fired only if the transaction is in an allowed state. This we ensure by an activity checking the state of the transaction, which was previously stored into a data store.
– When revoking a C-fact, after which a child transaction has been started, the child transaction must be completely revoked. This is done by calling a compensation throwing event by the revoke, followed by performing the compensation activity by the corresponding parent transaction.
– In the process flow, there can happen a situation, that a P-fact was already created (the P-act has been finished), while a revoke moves the process to a state preceding performing the P-act. In this case, it is necessary to "throw-away" the P-fact. We solve this using a BPMN compensation element and the respective compensation activity, similarly to the previous point.
– A revoke must be initiated by the actor who performed the respective C-act to be revoked. This is ensured by using the same identifier for the swimlane of the actor role initiating the revoke as for the actor role of the respective transaction.

A revoke works in the following steps according to the transaction pattern. First, the revoking actor asks the other actor for granting the revoke. The other actor allows or refuses. If the revoke is allowed, the main process returns to the appropriate state. We model this by using simple BPMN *subprocess* with a set of appropriate activities (fig. 7).

4.7 The Resulting BPMN Model

The complete transaction pattern described by the BPMN notation illustrates fig. 6[4]. Although it describes only one transaction, it is very complex and complicated. As it is presented in section 5 and discussed in section 6, models containing more than one transaction are not easily readable by usual readers and it is recommended to use them for the process execution in BPM systems.

4.8 The Execution

Apart from documentation purposes, BPMN models can be simulated and/or executed. While designing the conversion, we tried to make the resulting BPMN model precisely following the required behaviour. Unfortunately, the BPMN

[4] The authors are aware that this and the following models may not be legible in the printed version. We recommend obtaining the electronic (zoomable) version. Also, the source models may be downloaded from `https://ccmi.fit.cvut.cz/methodologies/bpmn/`

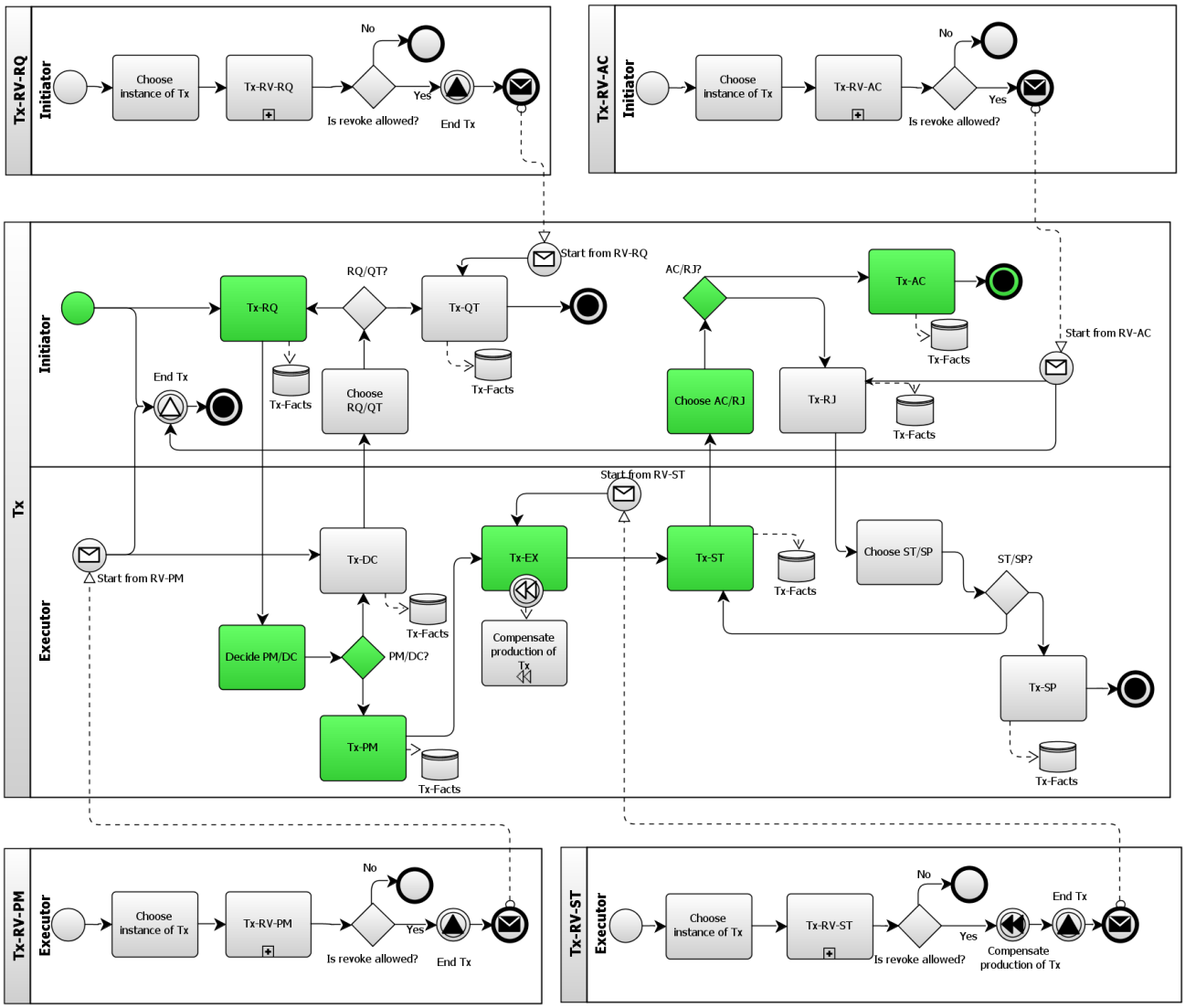


Fig. 6. Transaction in BPMN, Happy flow is marked by green colour

standard does not specify the execution implementation details. Each company developing BPM system (system for modelling, simulation and execution of processes), as Intalio, BizAgi or IBM, has their specific implementation, which requires various additional modelling and programming steps necessary to make the model executable. At the same time, some of the BPMN constructs may not be supported or they are implemented differently. All these aspects must be taken into consideration for turning the resulting BPMN models into an executable form. Generally, here are the things that must be implemented:

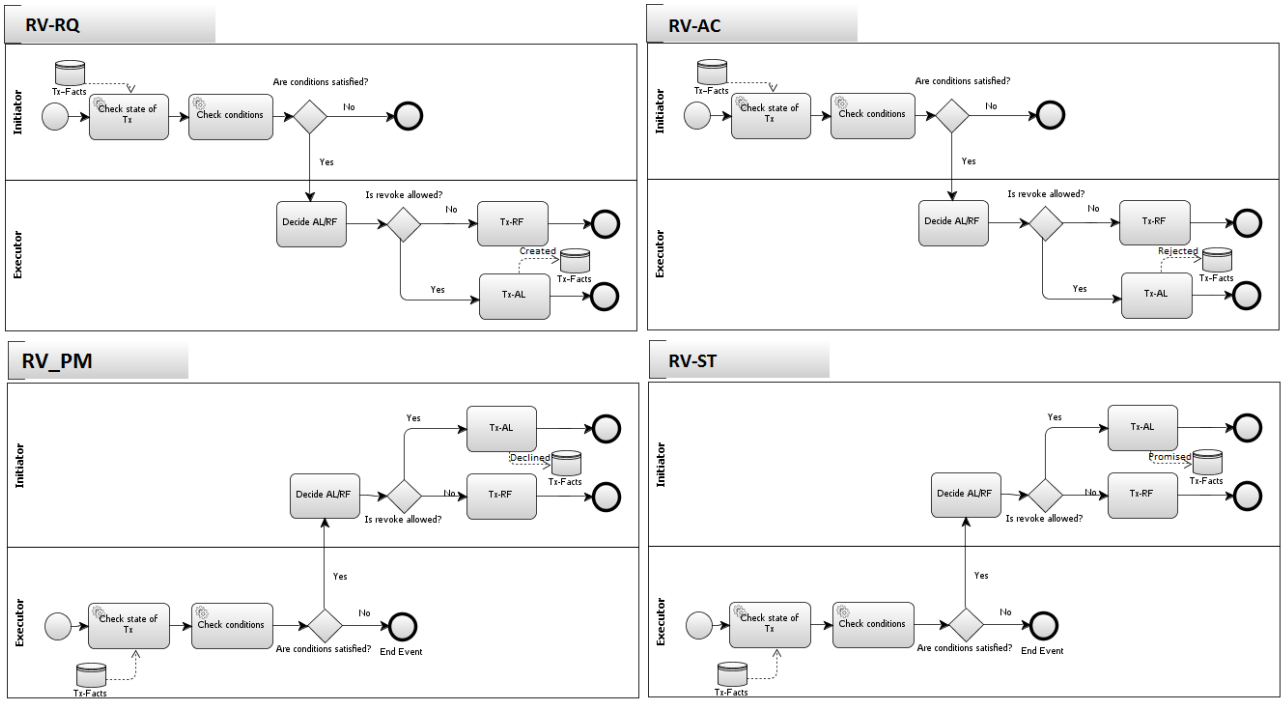


Fig. 7. Revokes in BPMN

- Agenda handling. The possibility to start a process and providing a "task inbox" of the required reactions on the originating C-facts. This requires developing some sort of user interface (UI).
- Allowing the participants to make their choices. Again, some sort of UI solves this. Also, some choices may be determined by complex facts evaluation specified in the Action Model. There are two possible approaches:
  1. Leaving the evaluation to users, which means the users have the rules in their head or consult the Action Model or any other codification of the rules.
  2. Programming the BPM system to (help) evaluate the rules. The extent to which the automation may happen depends on the possibilities of the BPM system used and also on the context (the availability of the necessary data in the company technological systems and their accessibility).
- Signals handling.
- Implementation of reading and writing data to data stores.
- Instance matching. Specific instances of transactions must be matched in some situations as child transactions (section 4.5) and revokes (section 4.6). Intalio and Oracle call this concept a "correlation".

5 Example – Case Voley

As an example for the demonstration of our method, the traditional Case Voley example [7] was selected because of its simplicity, yet including the substantial constructs. In fig. 8 and fig. 9 there are OCD and PSD diagrams of this example.

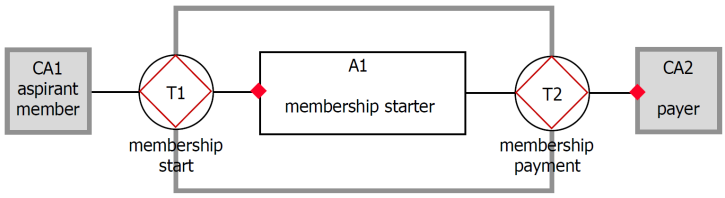


Fig. 8. OCD of Case Voley[7]

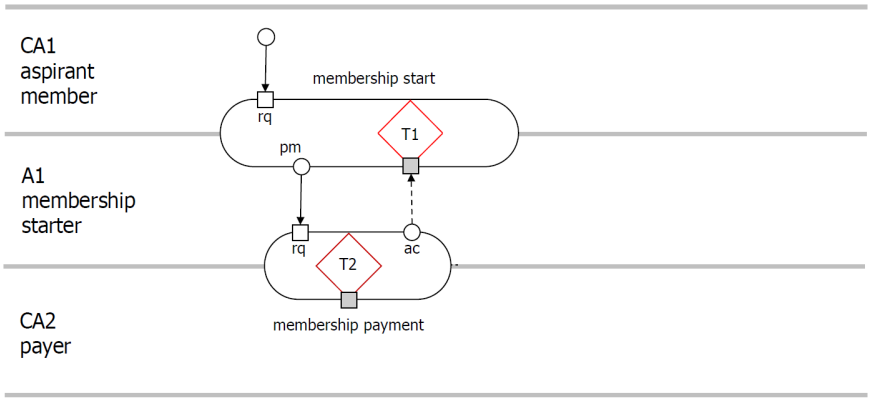


Fig. 9. PSD of Case Voley[7]

The process has two transactions and three actors. The transformed BPMN model converted by the described method is in fig. 10 and fig. 11 . Subprocesses depicted in fig. 7 are not shown here, as they are generally the same.

6 Discussion and Conclusions

The limitation of typical BPMN models from the view of the PSI-theory lie in their limited expression of reactions to unexpected situations. Many situations like decline, reject and especially revokes are not covered in the models, which causes operation troubles. The presented conversion method offers a remedy

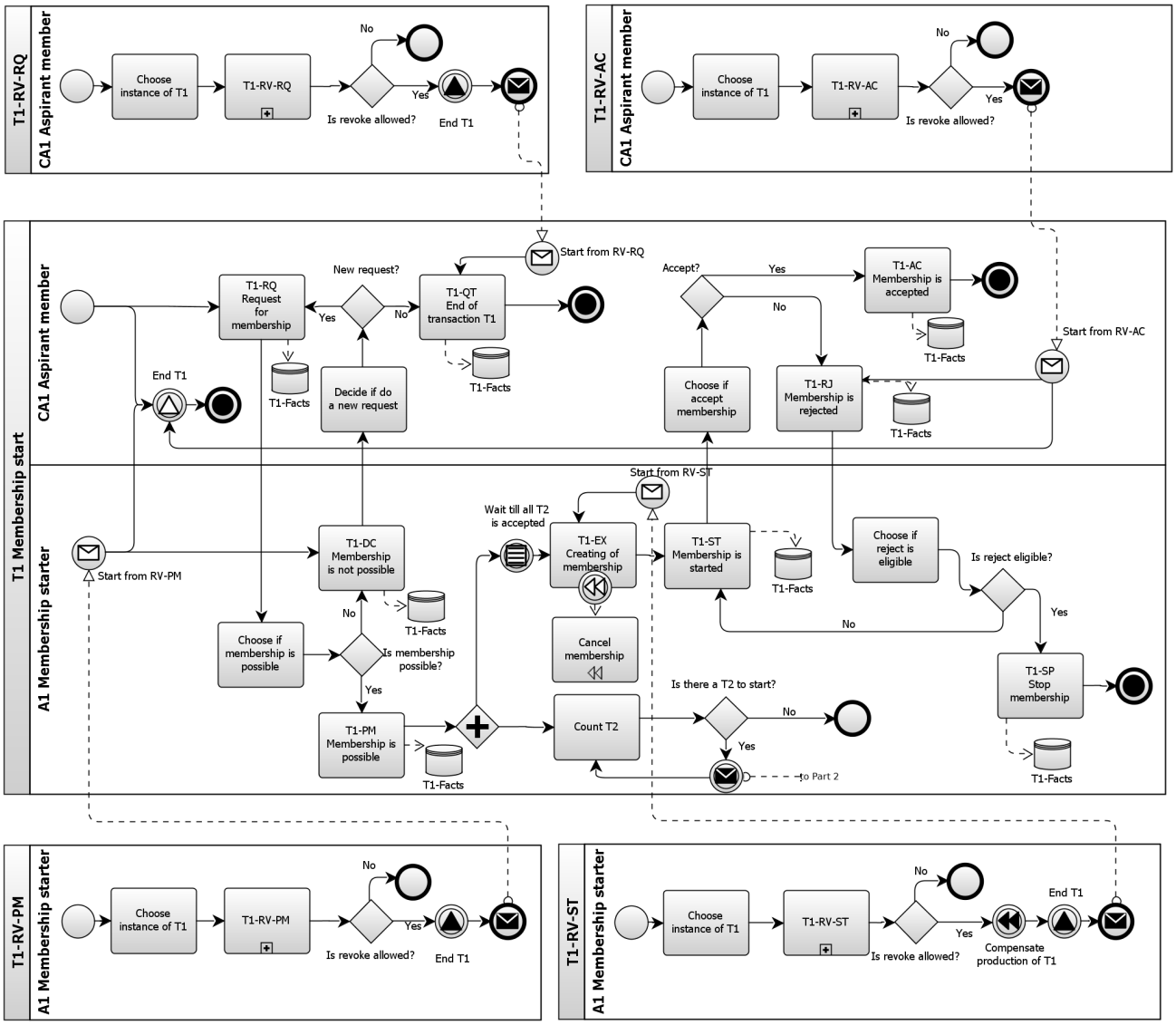


Fig. 10. Case Voley converted into BPMN – part 1

to this by bringing the *complete transaction pattern into BPMN*, which means including all revokes. Moreover, compared to the previous efforts, our method deals with spawning of *multiple child transaction instances* (initiation links with multiplicity $\neq 1$) and waiting for them in the parent transaction (waiting links with multiplicity $\neq 1$). Also, the resulting models are *executable*.

As for the DEMO models covered, the described conversion method covers the Construction Model plus the Process Model. Based on a concrete BPM system implementation, decision rules contained in the Action Model can be

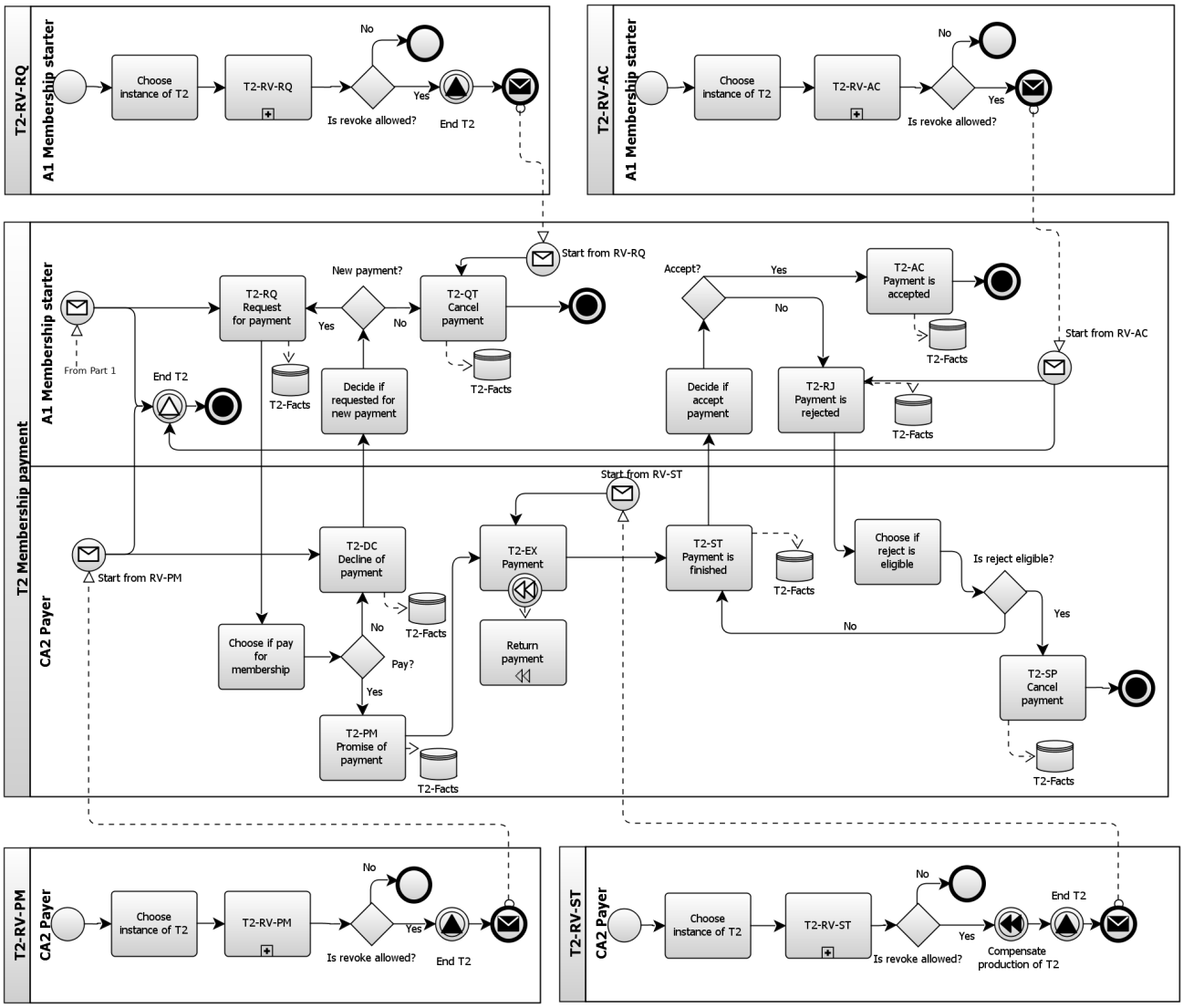


Fig. 11. Case Voley converted into BPMN – part 2

incorporated in the respective activities, as described in section 4.8, which is also true for rules from the State Model.

The concept of interstriction has not been discussed, however a keen reader has probably realised that whenever an actor in its activity needs a specific information from another transaction, it is simply modelled by accessing the respective transaction data store.

As DEMO models exhibit the C4E criteria [3], it is interesting to discuss them with the respect to the resulting BPMN models:

– Coherent – As we do not cover multiple types of models, this criterium is not applicable.

– Comprehensive – As we cover the complete transaction pattern, the resulting BPMN models exhibit the same comprehensibility as the CM + PM models and partially the AM and SM.
– Consistent – The hypothesis is that if the method is used properly (preferably by automation), the resulting BPMN models will be consistent, as bigger models are created from the presented basic building blocks using concatenation and recursion. However, this hypothesis must be explored and verified in future research.
– Concise – There is nothing superfluous in the resulting models. The reader may make a proof for themselves using proof by contradiction: trying to remove some of the generated elements results in malfunction.
– Essential – Not applicable for our method, as this quality is related to the distinction axiom, which we left out, as explained in section 3.

The example shows that in spite of the simplicity of the DEMO model involved, the resulting BPMN model is complex. The reason is mostly the complete transaction pattern, which covers all the possible situations according to the PSI-theory. The question arises about the human readability. There are several points to this topic:

1. In practice, the model may be made smaller by leaving out the parts, which are not applicable (which means they (almost) never happen). These are typically the revoke patterns.
2. Yet, for complex models the resulting size may remain still unmanageable. In this case it would be advisable to cut the model into smaller pieces using some sort of decomposition and/or *link* BPMN elements. The concrete way how to do this may be explored in a future research.
3. It is questionable whether a human readability is required. If one wants human-readable diagrams according to the PSI-theory, the DEMO diagrams are the solution, as they have been tailored to it. It may be the case that learning and applying them comes at a lower cost than forcing the diagrams into a BPMN notation, just because "BPMN is the standard".

Our stance is that the greatest possibilities of our method lie in *machine readability*, which means generating BPMN models that can be fed into a BPMN execution system to implement an automated workflow that is able to react to every possible situation specified by the complete transaction pattern, not just a typical BPMN "happy path with a bit of branching".

Apart from converting the DEMO models, the conversion may be applied also for analysis of existing BPMN models of business processes as described in section 2.1. The way of working would be to transform the BPMN models into DEMO and then generate the "supercharged" BPMN version by converting them back using our method.

As for the future work, a verification on a bigger models from practice is necessary. As such conversion will not be feasible by hand, an implementation of the conversion automation will be required.

Acknowledgements

This research has been funded by CTU SGS grant No. SGS16/120/OHK3/1T/18. The authors wish to deeply thank ForMetis BV and especially Dr. Steven van Kervel for the kind support of this research.

References

1. OMG: OMG: Business Process Model and Notation (BPMN) Version 2.0
2. Silver, B.: BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide: A structured approach for business process modeling and implementation using BPMN 2.0. Cody-Cassidy Press (October 2011) 00000.
3. Dietz, J.L.G.: Enterprise ontology: theory and methodology. Springer, Berlin; New York (2006) 00021.
4. Dietz, J.L.G., Hoogervorst, J.A.P., Albani, A., Aveiro, D., Babkin, E., Barjis, J., Caetano, A., Huysmans, P., Iijima, J., Kervel, S.J.V.: The discipline of enterprise engineering. International Journal of Organisational Design and Engineering **3**(1) (2013) 86–114 00042.
5. Op 't Land, M., Dietz, J.L.G.: Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations. In van der Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C., Albani, A., Aveiro, D., Barjis, J., eds.: Advances in Enterprise Engineering VI. Volume 110. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 77–92 00000.
6. Décosse, C., Molnar, W.A., Proper, H.A.: What Does DEMO Do? A Qualitative Analysis about DEMO in Practice: Founders, Modellers and Beneficiaries. In van der Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C., Aveiro, D., Tribolet, J., Gouveia, D., eds.: Advances in Enterprise Engineering VIII. Volume 174. Springer International Publishing, Cham (2014) 16–30 00000.
7. Dietz, J.L.: THE ESSENCE OF ORGANIZATION - AN INTRODUCTION TO ENTERPRISE ENGINEERING. Sapio bv (2012) 00000.
8. Guizzardi, G., Wagner, G.: Can BPMN be used for making simulation models? Lecture Notes in Business Information Processing **88 LNBIP** (2011) 100–115 00017.
9. Caetano, A., Assis, A., Borbinha, J., Tribolet, J.: An Application of the Ψ-Theory to the Analysis of Business Process Models. SpringerLink (2013) 258–267
10. Van Nuffel, D., Mulder, H., Van Kervel, S.: Enhancing the Formal Foundations of BPMN by Enterprise Ontology. In van der Aalst, W., Mylopoulos, J., Sadeh, N.M., Shaw, M.J., Szyperski, C., Albani, A., Barjis, J., Dietz, J.L.G., eds.: Advances in Enterprise Engineering III. Volume 34. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 115–129 00000.
11. Naplava, P., Pergl, R.: Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment. In: 2015 IEEE 17th Conference on Business Informatics. Volume 2. (July 2015) 18–26
12. Figueira, C., Aveiro, D.: A New Action Rule Syntax for DEmo MOdels Based Automatic worKflow procEss geneRation (DEMOBAKER). In Aveiro, D., Tribolet, J., Gouveia, D., eds.: Advances in Enterprise Engineering VIII. Number 174 in Lecture Notes in Business Information Processing. Springer International Publishing (May 2014) 46–60 00000.

13. Heller, S.: Usage of DEMO Methods for BPMN Models Creation. Master thesis, Czech Technical University in Prague. Computing and Information Centre. URL: https://ccmi.fit.cvut.cz/wp-content/uploads/2017/03/Heller_thesis_2016.pdf (2016)
14. Mráz, O.: Možnosti využití metodiky DEMO pro zvýšení kvality BPMN modelu. Master thesis, Czech Technical University in Prague. Computing and Information Centre. URL: https://ccmi.fit.cvut.cz/wp-content/uploads/2017/03/Mraz_thesis_2016.pdf (2016)

Chapter 18

Modelling and Prototyping of Business Applications Based on Multilevel Domain-Specific Language

[219] Pergl, R. Modelling and prototyping of business applications based on multilevel domain-specific language. Lecture Notes in Business Information Processing, volume 88 LNBIP, 2011: pp. 173–191.

Modelling and Prototyping of Business Applications based on Multilevel Domain-Specific Language

Robert Pergl

Department of Information Engineering,
Faculty of Economics and Management,
Czech University of Life Sciences,
Prague, Czech Republic
pergl@pef.czu.cz

Abstract. An effective approach to modelling and prototyping of business applications is presented in this paper. The approach is based on three concepts: The concept of data structure abstraction, the concept of a behavioural model based on the dynamic functional approach and a design approach based on creating multiple levels of a domain-specific language. The characteristics of each concept are presented. A technique how to combine them together to create highly detailed descriptive models that may be easily turned to prototypes is shown and demonstrated. Limitations are formulated and benefits over the object-oriented approach are discussed, as well.

Key words: Functional modelling, Prototyping, Domain-specific languages, Abstract data structures, Functional programming languages, Clojure, Lisp

1 Introduction

Some voices in the computer-science community start to point that computer science development is stuck in the object-oriented approach (OOA) and there is a several-year stagnation in the development of computer science techniques and paradigms. Experience shows that OOA has its considerable limits and that it has failed in may of its promises in practice ([9])[1]. New approaches are being called-for that may have a potential to boost flexibility and reusability of software systems. It seems that there is no "silver bullet" in the form of omnipotent approach or paradigm ([7]) and no new revolutionary paradigm is at stage. The development we may do today lies in combination of existing paradigms, some of them almost abandoned in spite of their potential. This is

[1] When we speak about object-orientation in this paper, we generally mean the pure object-orientation present e.g. in Smalltalk modelling language that is able to offer as much benefits as may be principally obtained. For a discussion how static and hybrid object-orientation spoils the OOA, see e.g. [7]

mostly an issue for systems development, however the modelling and prototyping paradigms are touched as well – models and prototypes paradigm should match the implementation paradigm.

We present three concepts in this paper that form the necessary foundation needed to develop a basic business application (BA) system model:

Data abstraction concept – We present a general abstract model for building data structures in section 2.

Behaviour abstraction concept – We present the concept of behavioural model based on a dynamic functional approach in section 3.

Architecture concept – We present a concept of multilevel domain-specific language approach in section 4.

We discuss each concept separately in each section and their combination for prototyping a practical task is shown in section 7.

2 Data Abstraction Concept

Generally, we may divide data structures into ([1])

- Primitive (numbers, characters, symbols, ...),
- Compound

The key data structures for complex models are the structures that aggregate other data structures, usually called data collections. There are three main classes[2] that take many different names:

Sequence type class – List, array, vector, sequence: the order of the elements is given by some rule (explicit order, alphabetical order, order based on the time of adding the element, etc.). Equal elements may occur multiple times. These collections are based on the mathematical term of n-tuple.

Set type class – This collection is based on the set as defined in mathematics. Each element may be present just once and elements have no order.

Associative type class – Associative map, associative list, dictionary: each element has a unique key by which it is identified. There is no order of elements.

Typically, various implementations are available in modelling languages (e.g. linked list vs. hash list vs. various trees) that differ in their (performance or memory consumption) characteristics. Various implementations are, however, not important for modelling and prototyping, moreover they may lead from the process of abstraction to optimisation too soon and bring unwanted complexity to the design process. As BA are in their nature very complex, we try to avoid any further sources of complexity in the early stages.

[2] The term "class" is used here in its general meaning, not the object-orientation semantics.

Attributes hold the data structures in the OOA. From this view, an object is an associative map where we access data element using its identifier[3]. In object-oriented models operations are encapsulated with the data structures.

3 Behaviour Abstraction Concept: Functional Paradigm

The functional paradigm is one of the "old" paradigms that have been almost forgotten in the realm of business applications. It used to be a domain of academic research or artificial intelligence since its beginnings in 1960s until last few years. However, in recent years (or even months), the functional approach experiences (a deserved) attention from both the community of professional programmers and computer scientists. Clean functional approach may offer many advantages both in the analysis, design and implementation.

3.1 Functional Model

A modelled system may be mathematically formalized as a mapping (function) from the set of inputs I into the set of outputs O. This provides the so-called "black-box" view on the system. To get a "white-box" view, it is necessary to specify the details of how the inputs are transformed to outputs. In the case of the *procedural (or imperative) model*, the specification consists of a sequence of **instructions**[4]. In the *functional model*, the specification consists of **functions**. Using a pure functional model provides some substantial advantages over the procedural model, which will be discussed in this section. The functional model, however, has its limitations, which will be discussed as well.

The procedural model achieves complex tasks by executing large numbers of instructions, which sequentially modify a system state until a desired result is achieved. Functional model achieve the same goal through composing nested functions, passing the result of one function as a parametre to another. All the data is handled through function parametres and return values (see Figure 1 compared to Figure 2). Those are so-called **pure functions**: they depend upon nothing but their parametres, and they do not perform any *side effects*, but just return a value. If we manage to model the system in a purely functional way, we get the following considerable benefits for implementation [16]:

Easiness to parallelise the algorithm : Since each function is a distinct, encapsulated unit, it does not matter if functions are run in the same process or even on the same machine.

High degree of encapsulation and reusability : Each function is effectively a black box. Therefore, understanding the inputs and the outputs makes understanding the function. There is no need to know or care about the inner details.

[3] Which is explicit in the JSON data format.

[4] The object-oriented model is a procedural one in this sense: message sending is performed in an imperative manner.

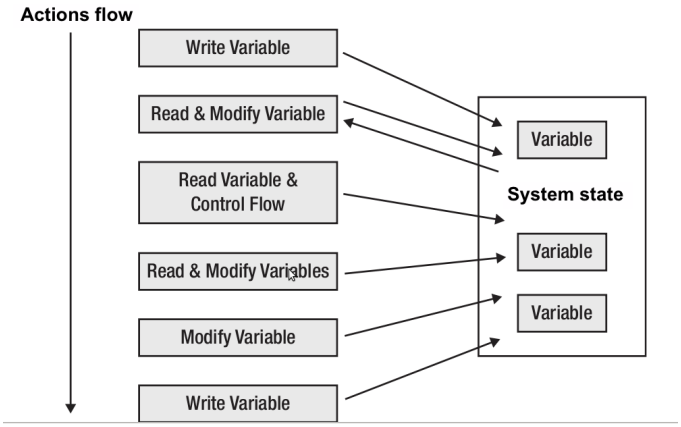


Fig. 1. Procedural model (taken from [16]).

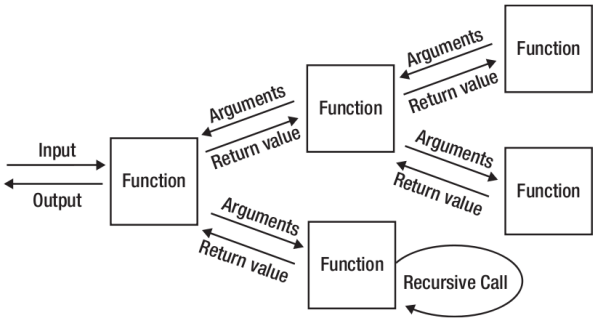


Fig. 2. Functional model (taken from [16]).

The OO model tries to achieve this through objects, but actually it is impossible to guarantee, because objects have their own state. An object's type and method signatures can never tell the whole story: it depends on how it manages its state and how its methods impact that state. In a complex system, this quickly grows in complexity and often the advantages of class encapsulation quickly disappear. Especially when class inheritance is used, which breaks encapsulation by exposing some implementation details (protected members, which non-private functions are called by which non-private functions, etc).

Easiness to reason about : In a purely functional model, the execution tree is very straightforward. By tracing the function call structure, it is clear what is happening. In order to understand a procedural, stateful model it is necessary not only to understand the code, but all of the possible permutations of state that may exist at any point in time.

Easiness to test : Pure functions are very easy to write unit tests for. One of the most difficult aspects of unit testing is anticipating and accounting for all the possible combinations of state and execution paths. Pure functions have well-defined, stateless behaviour that is extremely simple to test. In addition, data structures used are very simple and easy to construct in the place they are used without a need for creating mock-up classes and objects implementing given interfaces or instantiating objects and setting their inner state through series of calls of its methods.

In practice, we are not usually able to model the system as fully pure functional. We need to introduce imperative instructions for the following situations:

- We need to model *side effects* like disk reading and writing and user interactions.
- We need to model *order* of operations.
- We need to model *time* in the system.

> These are serious limitations, however in most business systems there is still much space for purely functional modelling that may be used for goal-oriented and constraint-based parts of the system. The key point is to limit the number and size of the parts of the system that have a procedural nature from the rest, that may be modelled functionally. This approach maximises the benefits mentioned above while not limiting the expressive power.

4 Architecture and Decomposition

4.1 Decomposition based on Fundamental Concepts

Decomposition is a famous technique that is used for engineering of complex systems. It means dividing a system into smaller subsystems in a tree-like manner. Thus complex systems are modelled by composing basic building blocks into compound structures, which again are composed into more complex ones. Each paradigm has its own system of building complex structures based on its fundamental concepts:

Structured model – procedures/functions
Object-oriented model – objects
Logic-based model – rules
Functional model – functions

> The decomposition makes it possible to focus just on a necessary level of detail in each single moment. However this plain decomposition does not work in practice as effectively as it seems. When a change needs to be done to a node, generally it brings the need to decompose the whole subtree again.

Unfortunately, a lot of changes happen at higher levels of the tree in the BA domain (new customer requirements or changes to the requirements), so this fact may cause intensive restructuring of the whole tree. This results in a non-linear complexity of performing model changes. Another characteristic of BAs is their evolution: new features are added over the time. The situation is similar: adding a new feature usually results in massive rewrites at all the levels of the tree.

Plain decomposition also makes the reusability difficult: it is theoretically possible to reuse the subtrees, however it is hard to find a subtree that may be reused, because every subtree was created only with the respect to the specific parent.

It is necessary to mention that today's most used approach, a decomposition into objects in the object-oriented approach accompanied with heavy encapsulation brings considerable problems as is shown e.g. in [2] and [7].

> Clean object-oriented approach makes changes and enhancements more flexible compared to structured decomposition, however its orientation on protocol and communication brings a considerable overhead.

For example many design patterns [8] try to deal with various problems trading flexibility in one area for another. Design patterns in the OOA exist because the only way to do any sort of abstraction in OO systems is to define communicating and cooperating objects ([7]).

> Decomposing problem to object classes implies creating mental model of objects dividing responsibilities they have. This model is built on abstractions the classes represent. However, the abstractions are created rather arbitrarily, especially in the beginning.

As the model changes, the responsibilities of the objects move, interfaces change and abstractions may become inconsistent with the mental model, making the model hard to understand. *Refactoring*[6] is then a common way to create change the model to match a new mental model and abstractions that we may not yet have. Warren Harris, a member of technical staff in the modelling Technology Department of Hewlett-Packard Laboratories, says to this: "... the very nature of the dynamics of OOP seems to be making these sorts of refactorizations" [7].

5 Motivation for a Domain-Specific Language Approach

Experience shows that OOA has its considerable limits and that it has failed in may of its promises in practice ([9]). New approaches are being called-for that may have a potential to boost flexibility and reusability of software models and systems. It seems that there is no "silver bullet" in the form of omnipotent approach or paradigm ([7]) and no new revolutionary paradigm is at stage. The development we may do today seems to lie in combination of existing paradigms. Let us present several motivational statements. Most of them are taken from [16], [2] and [7].

– "The truth of the dynamics/evolutionary situation is that you need all the code (and models! Note by the author) in front of you so that you can massage it into the new form you want it to take. ... Minimizing outside dependencies is a good thing, but doing this at the class level is usually just too fine grained." (Warren Harris in [7])
– OO approach encourages a high degree of ceremony and bloat. Simple functionality may require several interdependent classes to make a mental model match some "story" about the problem. Most of the bulk of a program is not actual program code, but defining elaborate structures to support it.
– Objects require creating communicating entities, which means that the system is modelled by building structures rather than by linguistic expression and description through form, and this often leads to a mismatch of language to problem domain.
– 16 out of 23 GoF fundamental design patterns [8] for OO languages are natural primitive constructs in functional languages due to avalability of:
  – First-class types (6): Abstract-Factory, Flyweight, Factory-Method, State, Proxy, Chain-Of-Responsibility
  – First-class functions (4): Command, Strategy, Template-Method, Visitor
  – Macros (2): Interpreter, Iterator
  – Method Combination (2): Mediator, Observer
  – Multimethods (1): Builder
  – Modules (1): Facade

5.1 Multilevel Domain-Specific Language Approach

A multilevel approach based on DSL [12] (ML-DSL) assumes that we have a domain-specific language available for modelling the system at each level of abstraction. Such a language layer is considered the perfect language for formalising a system[5].

> Using the multilevel domain-specific language approach means that we use this language at a certain level without having it specified in the lower level first.

This is the most important difference to the general Domain-Specific Language approach, where there is no strict segregation to levels specified. Multiple levels of the language brings advantages of the decomposition, while maintaining the expressiveness of the language (which is missing in plain top-down approach).

For ML-DSL we assume that the set of compound data structures consisting of three classes of data types mentioned in section 2 is available and that there are functions defined above them that provide the lowest level of the DSL for manipulating them. They form the leaves of the decomposition tree. For the

[5] In OO modelling this approach sometimes leads to the Facade design pattern, however the flexibility of it is limited in the aspect of malleability and extensibility of the pattern.

functional modelling and programming, they are the primitive functions for manipulating the basic data structures. Further on, we will call them **DSL-FL**, the domain-specific language of functional language.

The complete model should be consistent, i.e. every DSL term should be either the DSL-FL or it should be defined at some lower level. It is not necessary to strictly go from top to down as we do in our example below, however in the end the model should be complete.

6 Rapid Prototyping of Functional Models

We need to choose a suitable *language* for expressing the functional models.

> The Lisp-style functional programming languages are well-prepared for developing domain-specific languages [16]. The advantage of using a language with an interpreter is the possibility to develop prototypes that may be used for simulations and testing and even for the following implementation of the system.

We use the *Clojure* programming language ([10], [16], [4]) for such purposes, which is a new rapidly-developing very pure dynamic functional language that starts to get a considerable number of fans among programmers both from research and the commercial sector. Its syntax is built on the famous Lisp programming language, but it omits some complexity that has been notoriously causing bad dreams to programmers and on the other hand it adds some powerful data structures and abstract concepts not present in Lisp. All core language functions are side-effect free and immutability "by design" is emphasized over immutability "by convention" used in other Lisp dialects that did not prove sufficient.

Clojure is implemented upon the Java virtual machine and the integration with the Java libraries is almost seamless. The performance is reported to be (or can be easily made) the same as the hand-written Java code [10]. We decided to use Clojure for our demonstration because of its transparent syntax and powerful data structures.

In Clojure the notation for calling the function with arguments is (as in Lisp):

```
(function-symbol argument1 argument2 ...)
```

We may specify the transformation performed by a certain (sub)system at the highest level like

```
(output (transformation (input *input-data*)))
```

We will call this DSL level `Level 0`. We may now specify lower-level languages. During the prototyping process we will perform iteratively just a few operations:

– Design the DSL to operate the actual level of abstraction.

– Design necessary data structures to hold the data of the actual level.
– Design primitive DSLs to access and manipulate data structures of the actual level.

7 ML-DSL Prototyping Demonstration

We will present the approach on a simple example of a payments pre-processing system. The system is expected to perform the following transformation: it takes lines that represent payment records as input. Each record (line) has the following structure:

```
<Payment code> <Payment amount expression> <Payment type>,
```

e.g.

```
Electricity for the main building 12000 1000 Building costs
```

The *payment code* specifies the purpose of the payment. It may consist of small and capital letters and spaces. The *payment amount expression* in the form of a sequence of numbers separated by any number of spaces follows. The rest of the line specifies the *payment type* (or an account in the accounting terminology). The goal of the transformation is to sum the amounts for the payments with the same code and type, thus

```
Electricity for the main building 1000 13000 Facility costs
Salaries 150000 Staff costs
Electricity for the main building 1000 1500 500 Facility costs
Salaries 2000 2000 Staff costs
```

should produce:

```
Electricity for the main building 17000 Facility costs
Salaries 154000 Staff costs
```

Let us now demonstrate the ML-DSL prototyping of our example by implementing it in Clojure. We will show the implementation of the DSL at each level. To stay in the picture, the reader may refer to the Figure 3 during the reading. It depicts each level's DSL and the relations between the levels.

7.1 Level 0

As mentioned, this level is a trivial one. In our case, we may map the general concepts of input, transformation and output to our domain to get our level-0 DSL:

```
(output-payments (aggregate-payments (read-payments *input-data*)))
```

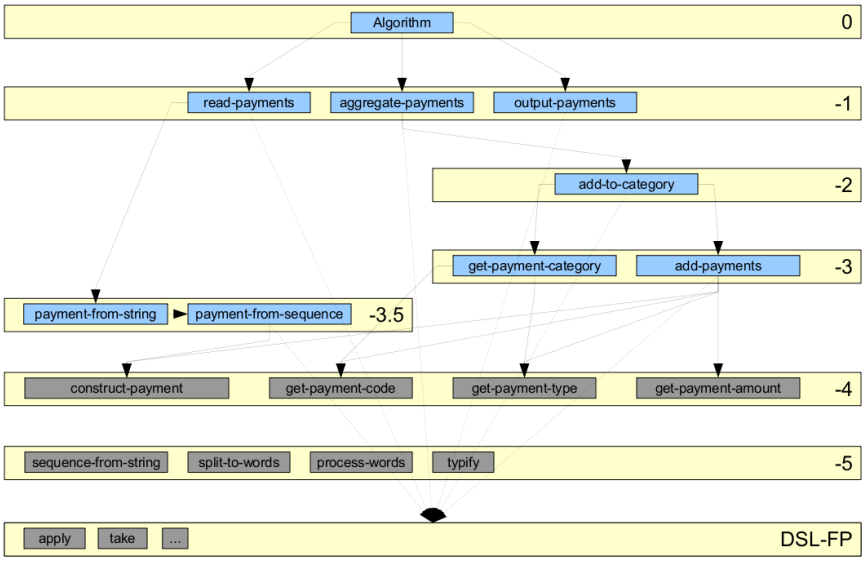


Fig. 3. The ML-DSL structure. Primitive functions are dark grey.

The read-payments function in our prototype will directly read the input data from a static binding:[6]

```
(def *input*
  ["Electricity for the main building 13000 Facility costs"
   "Salaries 150000 Staff costs"
   "Electricity for the main building 1000 1500 500 Facility costs"
   "Salaries 2000 2000 Staff costs"])
```

The square brackets denote a Clojure data type vector, which belongs to the class *sequence* according to the taxonomy in section 2[7]

7.2 Level −1

We dive one level down and we will use a lower-level DSL for working with the DSL of the higher level: functions `read-payments`, `aggregate-payments` and `output-payments`.

[6] In the system's implementation, it would read the data from some data source like a text file, a database or a network.

[7] Actually there is an abstract type called *sequence* in Clojure that abstracts sequential types like lists, vectors, strings. However this is a language detail that is not important for our purpose.

On the input, there is a sequence of strings representing payments. The reading function thus needs to perform a transformation on all the elements making *payments* out of strings.

```
(map function sequence)
```

> Clojure *map* function takes a function and a sequence. Its result is a sequence of elements transformed by the function.

Thus we may define the function:

```
(defn read-payments [lines]
  (map payment-from-string lines))
```

The `[lines]` expression has the meaning of input parametre to the function.
Let us define the transformation now:

```
(defn aggregate-payments [payments]
   (reduce add-to-category {} payments)))
```

The processing means to aggregate payments according to their category. The DSL we need here needs to express the fundamental operation, i.e adding the payment to an appropriate category. The `payments` input will be represented as a sequence of payments. The processing task is solved by aggregating all payments in the sequence. For this operation we use the Clojure `reduce` function.

```
(reduce reduce-function accumulator list)
```

> *Reduce* takes the starting accumulator and processes the elements of the list by feeding them to the reduce-function. The reduce-function takes the accumulator and an item and produces a new version of the accumulator in each iteration.

We decided to select Clojure map data because it will enable us to locate each payment category effectively by its key. The map data structure has the following syntax in Clojure:

```
{key value key value ...}
```

Keys and values may be of any type. We provide an empty map `{}` as the accumulator starting value.

The output function will in the end transform the map into a plain sequence of aggregated payment categories:

```
(defn output-payments [payments-map]
  (vals payments-map))
```

It uses the Clojure `vals` function to extract all values from the map. Despite the fact that it is trivial, we may start enjoying the REPL-style rapid prototyping.

REPL stands for "Read", "Evaluate", "Print", "Loop" which is typical for functional – especially Lisp-like – languages development environments. The interpreter continuously reads and evaluates the input, which makes building functional prototypes feel more responsive and enjoyable: we may play around with small pieces of code that are easy to debug and experiment with various ideas very fast.

"*When building something thats never been built before, the only way to get it right is to get it right every step of the way, by repairing anything that is new and broken as soon as its written and by making sure that nothing that is already ok becomes broken. In writing, this means proofreading each sentence as soon as its complete, if not sooner. When a paragraph is done, proofread it.*" (Richard P. Gabriel in [7])

More about REPL can be found e.g. in [14].
Let us suppose we have some result in the form

```
{"cat1" "payment1" "cat2" "payment2"}
```

We may evaluate `output-payments` in the REPL[8]:

```
=> (output-payments {"cat1" "payment1" "cat2" "payment2"})
["payment1" "payment2"]
```

Please notice that we still use the most general concept of *payment*, ignoring even the simplest notions how it may look like! This is a luxury of simplicity that is generally impossible in object-oriented / statically typed environments.

7.3 Level −2

We will implement adding to category required by the upper level now. This function takes a payments map that consists of payments keys and corresponding payments and returns a new map, where the payment has been added to the appropriate category. We need two functions for our DSL at this level: `get-payment-category` that will provide us a unique key for a payment and `add-payments` that will merge two payments into one merging whatever needs to be merged. From the DSL-FL we will need the operation of getting the value of the given key from a map and associating a value to the key in the map.

Getting the value of a key is trivial in Clojure: calling the map as a function with the parametre that is the key.

```
(map key)
```

[8] We will depict evaluating an expression in the REPL by "=>" and provide the result of evaluation on the following line.

The associating is done with the `assoc` function:

```
(assoc map key value)
```

Thus our function will be:

```
(defn add-to-category [payments-map payment]
  (let [key (get-payment-category payment)]
    (assoc payments-map key (add-payments payment (payments-map key)))))
```

> The *let* form enables to bind expressions to comprehensible names[9]. It takes two parametres: a vector with pairs "symbol-expression" and the body that uses these symbols.

We may test our DSL prototype in practice now. We may invent our own mock-up data structures that will be accessed by those two functions to develop and test the functionality of this level. The simplest data structure we may specify for this purpose is a payment consisting of two values: its category and its value: e.g. `["cat1" 3000]`. We may now write trivial functions for working with this data structure:

```
(defn get-payment-category [payment] (first payment))

(defn add-payments [payment1 payment2]
  (if (nil? payment2)
    payment1
    [(first payment1) (+ (second payment1) (second payment2))]))
```

If `payment2` is nil, the result is `payment1` unchanged. Otherwise, the result is a new payment constructed from the code being the first element of `payment1` and the sum of payment amounts being the second elements[10]. These functions may be immediately defined and tested in the REPL, however we will not do this, for they are really trivial. These functions will be redefined by real versions later when we deal with the lower DSL level.

Now we have a complete DSL for this level and we may test it:

```
=> (add-to-category {"cat1" ["cat1" 3000]} ["cat2" 500])
{"cat2" ["cat2" 500], "cat1" ["cat1" 3000]}

=> (add-to-category
     {"cat2" ["cat2" 500], "cat1" ["cat1" 3000]}
     ["cat2" 50])
{"cat2" ["cat2" 550], "cat1" ["cat1" 3000]}
```

[10] Please notice the Lisp-style prefix notation of addition first the function name, then the parametres.

7.4 Level −3

In this level, we will redefine the functions that are used in the higher-level DSL using "ideal" lower-level DSL functions:

```
(defn get-payment-category [payment]
  [(get-payment-code payment) (get-payment-type payment)])

(defn add-payments [p1 p2]
  (if (nil? p2)
    p1
    (construct-payment (get-payment-code p1)
      (+ (get-payment-amount p1) (get-payment-amount p2))
      (get-payment-type p1))))
```

7.5 Level −3.5

Before dealing with concrete representation of the payment let us implement the function needed in Level -1 `payment-from-string`. We abstracted payment as a sequence individual payment elements, thus we will construct it from a sequence:

```
(defn payment-from-string [string]
  (payment-from-sequence (sequence-from-string string)))
```

`payment-from-sequence` is trivial: we just need to take 3 first elements from the sequence:

```
(defn payment-from-sequence [seq]
  (apply construct-payment (take 3 seq)))
```

Which we may test in the REPL:

```
=> (payment-from-sequence ["a" 3 "x"])
["a" 3 "x"]
```

We denoted this level as −3.5 because it has a more local (specific) purpose.

7.6 Level −4

We will implement the DSL used in level -3.

> This is the level where we first need to consider the representation of the data structure of *payment*. It implies that changing the structure of payment will not affect levels above −4!

```
(defn get-payment-code [[code amount type]]
  code)
```

```
(defn get-payment-amount [[code amount type]]
  amount)

(defn get-payment-type [[code amount type]]
  type)
```

These functions are analogous to *property getters* in object-oriented languages. We used the concept called "destructuring" here, which provides a highly expressive mechanism [3]. The argument passed to `get-payment-*` functions is the payment vector. However, we directly specify its structure in the parametre. This results in destructuring the elements to the appropriate bindings.

Thus `construct-payment` will now look like:

```
(defn construct-payment [code amount type]
  [code amount type])
```

This function is analogous to *object constructors* in object-oriented languages.

7.7 Level −5

Now we descend into a pure internal level that maps the specific input format into our higher-level sequence. To make the message of Figure 3 as clear as possible, we did not depict details of this level into it.

> We may implement this level using any approach and apply various implementation tricks here, because this is the only level that will change if the input format changes (which we consider very probable). All higher levels will remain intact.

`sequence-from-string` may be achieved by the transformation chain `string -> split-to-words -> process-words` and performing merging of mergeable elements, i.e. amounts that should be summed:

```
(defn sequence-from-string [string]
  (reduce merge-item [] (process-words (split-to-words string))))
```

Splitting to words may be achieved simply by Clojure `re-seq` function that takes a regular expression (denoted as `#"exp"` in Clojure) and a string and returns a sequence of successive matches of pattern in string:

```
(defn split-to-words [string]
  (re-seq #"[a-zA-Z_]+|[0-9]+" string))
```

In REPL this may be tested like:

```
=> (split-to-words "Electricity for the main building
                    1000 1500 500 Facility costs")
("Electricity" "for" "the" "main" "building"
 "1000" "1500" "500" "Facility" "costs")
```

Processing the words consists of converting each element to a proper type:

```
(defn process-words [seq-of-strings]
  (map typify seq-of-strings))
```

The typify function should convert numbers to number type and leave strings intact. We may use a Java class method `parseInt` of the class `Integer` to turn strings into numbers. Java class methods (as well as instance methods) may be directly called from Clojure, like

```
=> (Integer/parseInt "4000")
4000
```

thus allowing us to define the `typify` function:

```
(defn typify [argument]
  (if (not (nil? (re-matches #"[0-9]+" argument)))
    (Integer/parseInt argument)
    argument))
```

The `typify` function matches the expression against a regular expression specifying whole positive numbers. If the argument matches the regexp, the parsed integer is returned, otherwise unchanged argument is returned.

Merging of the items into an accumulator is implemented in the functional style:

- If the accumulator is empty, the result is a new list with the item.
- If the last element of the accumulator is string and the item is a string, then the last element of the accumulator is replaced by concatenation of this element, the space and the item.
- If the last element of the accumulator is number and the item is a number, then the last element of the accumulator is replaced by the sum of the this element and the item.
- Otherwise just append the item to the accumulator without any processing.

We provide the complete code together with "test-cases" in comments (denoted by ";;"):

```
(defn merge-item [accu item]
  (cond
    ;; (merge-item [] "b") -> ["b"]
    (empty? accu) [item]
    ;; (merge-item ["a"] "b") -> ["a b"]
    (and (string? (last accu)) (string? item))
      (conj (pop accu) (str (last accu) " " item))
    ;; (merge-item [9] 3) -> [12]
    (and (number? (last accu)) (number? item))
      (conj (pop accu) (+ (last accu) item))
    ;; (merge-item [9] "b") -> [9 "b"]
    :else (conj accu item)))
```

This is relatively a little bit more complex piece of an algorithm. We omit the detailed explanation of the Clojure DSL-FL used here, we present it here to emphasise the fact that this processing resides in the lowest level, so its complexity is hidden to higher levels and may be safely fine-tuned and optimised as needed.

8 Discussion

The presented prototype solution consists mostly of the so-called "one liners", i.e. very short functions that can be easily comprehended, debuged and easily reused.

By combing the ML-DSL approach with the functional concepts, we may achieve the following qualities of a prototype:

- The prototype is easy to read and comprehend: Lower levels work with abstract terms without knowing anything about the underlying data structure.
- Side-effect-free blocks have much simpler lexically-bound context easier to isolate and comprehend.
- The prototype is easy to debug: we build simple blocks upon each other; side-effect free blocks can be tested without complicated set-up of the environment state and interacting objects (as in the object-oriented approach) and for the same input it gives the same output.
- The prototype achieves flexibility with the respect to input data structures changes: if this happens, we just modify the few functions at a given level that access the data structure. All the higher-level functions will remain intact.

One of the greatest benefits of the ML-DSL stated in section 4 is elimination of redesign of subtrees due to low-level format change or new feature implementation. We may demonstrate this situation on our example by updating the input format to support "+" as a word that can be interpreted either as adding two surrounding numbers or as a part of a string. Although this requirement makes the input format more vague as "+" can appear in different context (as a word or as an operator), this change would require just an incremental update of the code in the lowest level:

We will change `split-to-words` to detect the `+` character as a word character:

```
(defn split-to-words [string]
  (re-seq #"[a-zA-Z_+]+|[0-9]+" string))
```

and we will enhance the `merge-item` function by a new condition branch[11]:

```
(and
  (number? item)
```

[11] The Clojure DSL-FL allows any number of predicates as arguments to the `and` function

```
    (= "+" (last accu))
    (not (empty? (pop accu)))
    (number? (last (pop accu))))
  (conj (pop (pop accu)) (+ (last (pop accu)) item))
```

All higher levels will remain intact as they work with abstract DSL independent on input data structure.

9 Summary and Conclusions

Functional analysis, design and programming of business applications offers an interesting alternative to the object-oriented approach. Its simple and yet powerful paradigm based on pure functions provides considerable benefits in situations where we do not need to model states, time-based behaviour and side effects like user inputs and outputs. The benefits may be achieved also by mere separation of the functional and procedural parts of the system.

The combination of functional approach, expressive data structures and a suitable syntax is a basis of the presented Multilevel Domain-Specific Language (ML-DSL) approach to modelling and prototyping. This approach is based on decomposition of the system model into layers of domain-specific languages, where each layer may be used without knowing the details of the lower layers. This makes possible to build expressive descriptions of systems and algorithms independently on the underlying data structures. Moreover, changes made to the lower levels do not affect the higher levels, which contributes to prototypes' flexibility. Thus this approach may be also recommended for use together with Agile methodologies, where flexible management of deliverables is needed [13].

The ML-DLS approach was demonstrated on a case-study of a simple payments-processing system. A ML-DSL prototype implementation was shown using the Clojure programming language that offers a Lisp-style effective syntax on the Java platform together with powerful data types and other modern features. The benefits of using the REPL (Read-Eval-Print-Loop) environment were also demonstrated: a live, responsive development where pieces of the DSL may be tested and demonstrated before building more complex prototypes.

When the Clojure programming language is selected also as an implementation language[12] a very effective modelling-prototyping-implementation development cycle is achieved, where semantic gaps between the phases almost vanish.

Modelling a system using the expressive ML-DSL approach provides a very precise system model (or directly a working prototype), but it takes quite a lot of effort and skills and it may be thus recommended to analysts skilled in abstraction and formal notations. Moreover, the functional approach takes a lot of practice and "unlearning" for those trained exclusively in procedural thinking, which is today still the majority of analysts and programmers.

[12] In spite of its syntax and paradigm quite unusual for today's programmers, Clojure starts to be used in enterprise-grade programming, see e.g.[11]

Acknowledgements

This contribution was elaborated with a support of grant no. 201011130035 of Grant Agency of The Faculty of Economics and Management of the Czech University of Life Sciences in Prague.

References

1. A.V. Aho, J.D. Ullman, J.E. Hopcroft, *"Data Structures and Algorithms"*, Addison Wesley (1983).
2. M. Ben-Ari, "Objects Never? Well, Hardly Ever!", In: *Communications of the ACM*, vol. 53, no. 09, pp. 32-35, ACM (2010).
3. M. Fogus, *Clojure Mini-Languages*, `http://blog.fogus.me/2010/03/23/clojures-mini-languages` (2010).
4. M. Fogus, Ch. Houser, *"The Joy of Clojure: Thinking the Clojure Way"*, Manning Publications (2010).
5. E. Gamma, R. Helm, et al., *"Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison-Wesley Professional (1994)
6. M. Fowler, K. Beck, *"Refactoring: Improving the Design of Existing Code"*, Addison-Wesley Professional (1999).
7. R. Gabriel, *"Objects have failed: Notes for a Debate"*, `http://www.dreamsongs.com/NewFiles/ObjectsHaveFailed.pdf` (2002).
8. E. Gamma, R. Helm, et al., *"Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison-Wesley Professional (1994).
9. I. Hadar, U. Leron, "How intuitive is object-oriented design?", In: *Communications of the ACM*, vol. 51, no. 5, pp. 41-46, ACM (2008).
10. S. Halloway, *"Programming Clojure"*, Pragmatic Bookshelf (2009).
11. S. Halloway, *"Clojure in the Field"*, `http://www.infoq.com/presentations/Clojure-in-the-Field` (2010).
12. M. Mernik, J. Heering, and A.M. Sloane, "When and How to Develop Domain-Specific Languages", In: *ACM Computing Surveys*, vol. 37, no. 4, pp. 316-344, ACM (2005).
13. Molhanec, M. "Towards an Agile Project Management in Product Design". In 32ND INTERNATIONAL SPRING SEMINAR ON ELECTRONICS TECHNOLOGY, 682-685. International Spring Seminar on Electronics Technology, ISSE. ISBN 978-1-4244-4260-7 (2009).
14. P. Norvig, *"Paradigms of Artificial Intelligence modelling: Case Studies in Common Lisp"*, Morgan Kaufmann (1991).
15. S.M. Ralph et al., *"Principles of Information Systems, Sixth Edition"*, Thomson Learning (2003).
16. L. VanderHart, S. Sierra, *"Practical Clojure"*, Apress (2010).

Chapter 19

Analysing Functional Paradigm Concepts: The JavaScript Case

[218] Janeček, L.; Pergl, R. Analysing Functional Paradigm Concepts: The JavaScript Case. In Recent Advances in Information Systems and Technologies, Advances in Intelligent Systems and Computing, Springer, Cham, Apr. 2017, ISBN 978-3-319-56534-7, pp. 882–891.

Analysing Functional Paradigm Concepts
The JavaScript Case

Lukáš Janeček, Robert Pergl

Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic,
{janeclu1,perglr}@fit.cvut.cz

Abstract. Hundreds of programming languages are available today and new ones are still emerging. Nevertheless, they are founded in several (old) paradigms. Knowing the essence of paradigms helps to orient oneself in this Babylon, which is challenging especially for the growing community of programmers with no computer science background. In this paper we focus on functional paradigm, which has a raising attention both in new languages (like Clojure and ClojureScript) and a growing support in traditional languages (like C++ and Java). We do not discuss why this happens here, but we focus on analysing fundamental concepts in the functional paradigm and functional programming languages. We describe them and divide them into two categories: key principles and additional principles. Next, we apply this conceptual framework to analyse the ES5 and ES6 versions of JavaScript. We conclude that ES6 is a good step towards functional principles support. Also, the presented conceptual framework may be used for similar analyses of other languages.

Keywords: functional programming, lambda calculus, JavaScript, ECMAScript 5, ECMAScript 6

1 Introduction and Motivation

There is an astounding number of programming languages available. For example, in the Wikipedia, there is a list more than 700 programming languages [1]. New ones emerge almost every year [1]. However, they are based on a few programming paradigms, being the imperative (or procedural) paradigm, the object-oriented paradigm, logic-based (or rule-based) and the functional paradigm (or applicative) [2]. Programming languages are based on one or more of the paradigms, which they embrace in their own style. There is a growing number of non-professional programmers, e.g. scientists from various fields, who lack the formal computer science education. This Babylon of languages becomes very confusing for them. In this paper we focus on the functional paradigm, which is very old, at the same, it gains popularity nowadays: new languages emerge (like Clojure [3]) and other languages adopt its concepts (C++, Java) [4], while the functional paradigm is strongly rooted in most of the today's mainstream languages, as well (Python, C#, F#, Ruby, Smalltalk and others). This

situation is in a strong contrast to awareness of the programmers, where most of them remain with traditional Fortran-style programming (iteration, if-then-else). Our goal is to present the key concepts in the functional paradigm to help programmers see the essence in programming languages.

2 Methodology

We identify key principles from the literature and explain them. We denote the principles by identifiers for easy referencing. We demonstrate their implementation in the popular JavaScript language – the ES5 and ES6 versions of the language. The result are specific conclusions about the functional principles support in JavaScript, as well as a general conceptual framework that can be similarly used for analysing other functional programming languages and languages with functional features. We do not dive deeply into explaining the benefits of using these concepts, nor discussion of their appropriateness for various situations. This discussion may be found in the references provided.

3 Key Principles of Functional Programming

The formal foundation of functional programming is the Lambda calculus, also written as λ-calculus. It was formulated by mathematician Alonzo Church in the 1930s as part of an investigation into the foundations of mathematics [5]. The Lambda calculus provides a simple semantics for computation, enabling properties of computation to be studied formally. We do not discuss these formal aspects of the Lambda calculus here, but we focus on the programming perspective.

The most fundamental principle is the notion of **first-class functions (P1)**. It is an essence of functional programming that functions are first-class citizens that may be manipulated as data. Since functions are considered values in their own right, it is natural for them to appear as arguments or results of other functions. Functions that takes other functions as arguments or that return functions as results are said to be **higher order**, and we refer to them as "functionals" to distinguish them from ordinary functions [6]. Functions may be named or *anonymous*, which corresponds to the notion of a Lambda function. Higher-order functions can be used for example in *functors*, where functionals are mapped over a structure, which provides a "better" alternative to iteration [7].

The **referential transparency (P2)** states that function call can be replaced by its return value (obtained by calling the function with the same arguments). It makes the order and count of execution irrelevant. This can be done with so-called *pure functions*. This means that function can not impose *side effects*, like a mathematical function. An example of a side effect is printing out some message, waiting for an input, sending something through the network, or only accessing some variable outside the function scope (for example global

variable or variable from parent scope). This poses severe limitations, however abiding the referential transparency has the following benefits [8]:

- Purely functions are remarkably easy to parallelize.
- Pure functions lead to a high degree of code encapsulation and reusability.
- They are easier to reason about.
- Pure functions are very easy to write unit tests for.

The referential transparency is tightly bound with **the immutability of variables and values (P3)**. This means, that if we assign a value to some variable, we cannot reassign it. This variable will hold this value until the program stops. From this perspective, variables are more like identifiers for data values, than slots for some changing data. In some languages (like Haskell: [9]), immutability is strictly embedded in the language. However, in languages with imperative features, the situation is more complicated and immutability must be explicitly managed. It can be achieved in several dimensions. The first step is disabling reassigning another value to an already assigned variable. This is usually achieved by a keyword (`const`, `val`, `final`, etc.). The effect is that we cannot assign a new value to this variable, but we still can change the value itself. For example, when we define immutable (or constant) variable named `user1` and value of this `user1` will be the user with name "George", we can not set `user1` to another user later, but we still can change the name of "George" to "John". Of course, this problem does not hold for values like integers or strings, just the compound types. The solution of this problem is to define immutable variables transitively. In this case, user object will have all properties defined as constants, so we can not change its name to "John". In case that user contains another compound object as its property, this object have to be also immutable (contains only immutable properties). This leads us to a question of standard libraries. To support immutability, types defined in libraries must be immutable and functions in these libraries must support operations with immutable structures. These functions have to return a new object with updated values instead of changing the existing object. For example, a sort function is not allowed to change array to sorted array, but it has to leave the original array untouched and return a new sorted array.

A **closure (P4)** is simply a pairing of a function with its environment: the bound variables that it can see [10]. It means, that function can access values from the enclosing block (lexical scope). More specifically, not from the context in which they are called, but from the context in which are defined. Closures are used for making the code more clear and readable. They also provide encapsulation (like private members in the traditional object-oriented programming).

A **recursion (P5)** is used in functional programming instead of loops. It is a situation when a function calls itself. There is a special type of recursion called tail-recursion. A tail-recursive clause is a recursive clause of the form

$$p :- q_1, .., q_n, p,$$

where $n \geq 0$, i.e. the last the last call in the body is a recursive call to itself. It is well known that tail-recursion can be replaced by iteration. This is because

there are no more calls after the tail-recursive one, which means that its binding environment can, with a bit of care, be discarded and the space reused [11]. Thus, if a compiler supports tail recursion optimisation, it solves the stack overflow risk of recursion in case of tail recursion.

3.1 Additional properties

Apart from the fundamental principles, we identified the following additional principles. Some of them also appear in other paradigms, like object-oriented programming or logic programming.

Lazy evaluation (P6) is a situation when a value of an expression is not calculated in the moment of declaration, but it is delayed until needed; It may also happen that the value is not evaluated ever [9]. The lazy evaluation is typically bound to pure functions, as side effects complicate the situation by the possibility that they may not be evaluated [12]. Lazy evaluation does not make much difference for atomic values, apart from possible small optimisation. However, their importance is substantial for collections. Lazy collections (usually lists) are a very common pattern for solving problems in functional style. They bring a possibility to work with virtually infinite streams by evaluating just the portions that are accessed [13], [14], [15].

Currying (P7) is another important additional principle of functional programming. Currying is a technique of transforming a function of multiple arguments into evaluating a sequence of functions, each taking one argument [16]. In fact, currying is the default mechanism in the Lambda calculus, while multiple-argument functions are technically just a "syntactic sugar". The same is true for the Haskell programming language, while most of other languages default to multiple arguments and use currying mostly to achieve *partially applied functions*. It is a situation when we pass fewer arguments to a function that the function expects [10]. It is a powerful abstraction mechanism enabling creation of specialised versions of functions. [10]

Pattern matching (P8) is a well-known concept not limited to the functional programming. Pattern matching provides the means to inspect and decompose nested data structures in a single statement [17]. Using this construct, a programmer can define different behaviour of a function based on distinct values and data structures without writing if-then-else constructs, which makes it a device of polymorphism.

Polymorphism (P9) [18] is a powerful abstraction principle, again not only limited to functional programming, it is also one of corner stones of object-oriented programming). *Polymorphic functions* are functions whose operands (actual parameters) can be of more than one single type. *Polymorphic types* are types whose operations are applicable to values of more than one single type [19].

To sum up, a proper functional programming language should implement the concept of functions as first class values, support closures, immutable variables and pure referential-transparent functions. Recursion is generally supported in

all today's languages including the imperative ones, however tail recursion optimisation is a welcome asset from the implementation perspective.

4 FP Analysis of JavaScript

Let us now explain how the identified principles are embodied in the JavaScript language (JS), arguably the most important language of the web [20]. JavaScript is designed as a dynamically typed scripting language. The current widely used edition is the ECMAScript 5^{th} Edition (ES5)[21] and a new standard ECMAScript 6^{th} edition (ES6) is available [22]. ES6 contains more direct support for functional programming constructs, but it is not fully supported in the current web browsers (in November 2016). The current adoption status may be checked in [23]. Currently, the code in ES6 for browsers is usually translated into the ES5 code for compatibility reasons.

4.1 ES5

ES5 is a shortcut for EcmaScript 5 from 2009 JavaScript standard [21]. In JavaScript, functions are first-class objects and can be assigned to variables, passed as function arguments or returned as a function result. Anonymous (Lambda) functions are supported in form of:

```
function(arguments,...){ return ___;}
```

We may conclude that principle (P1): first-class functions is supported in JS. Functors are supported with arrays, but there is a small amount of standard functions based on them. They can be supported using libraries like Lodash [24] or Underscore [25].

Principle (P2): referential transparency is not guaranteed nor managed, as functions are not required to be pure and also (P3): immutability is not directly supported. There is no syntax for specifying immutable variables and no functions or objects from standard library that would provide support for immutability. So when we write:

```
var a = [5,2,4,3];
a.sort();
console.log(a); // [2, 3, 4, 5]
```

The value of variable a will be [2, 3, 4, 5]. So the function `sort` sorts the original array instead of returning the sorted array as its return value and letting the original array unchanged. (In fact, this function returns the sorted array as a return value, nevertheless it sorts the original array anyway.) The only way, how to declare a (local) variable is using the keyword `var`[1], and variables can be reassigned. For example, this is a valid code:

[1] A variable can be also declared without the `var` keyword, which makes the variable global.

```
var a = 10;
a = 20;
console.log(a) // 20
```

There is immutable support for properties in objects:

```
var obj = {};
Object.defineProperty(obj, 'key', {
  configurable: false,
  writable: false,
  value: 'some data'
});
```

When the `writable` attribute for a property is set to false, any attempt to change the value of the property fails [21].

Object `obj` now contains property `key` with value `some data`. This property can not be changed (`writable: false`) or deleted (`configurable: false`). The second option is freezing an existing object:

```
var obj = {key: 'value 1'}
console.log(obj.key) // value 1
obj.key = 'value 2'
console.log(obj.key) // value 2
Object.freeze(obj)
console.log(obj.key) // value 2
obj.key = 'value 3'
console.log(obj.key) // value 2
```

Because of standard functions which mutate data, most of the code in JS is not referential transparent – (P2) is not supported. There is a possibility to mitigate this situation by using libraries providing basic support for immutable structures and functions manipulating these structures (for example `immutable.js` [26]).

Functions are scope for variables. Objects may be created from JavaScript functions. Properties of these objects can be accessed using closures. Because these properties can be also changed, using closures breaks the referential transparency – If we invoke closure C, then change a variable that is used in this closure and then invoke C for the second time, the result of this closure may be different.

Nevertheless, closures (P4) are present and provide a powerful mechanism, which is leveraged in introducing the concept of modules, which is missing in the language. Modules are implemented in the AMD library ([27]) by enclosing the exported functions in another function, thus forming a closure and providing encapsulation.

Principle (P5): recursion may be used in JS, however tail call optimization is not present. Functions that recurse very deeply can fail by exhausting the return stack [20].

As for the additional principles, (P6): lazy values are not supported, (P7): currying is not supported in the language, but it can be achieved indirectly [28] or using libraries [29]). (P8): pattern matching is not present.

(P9): Polymorphism is supported by a *prototype inheritance* [30] in a rather object-oriented fashion. Polymorphism in JavaScript deserves a deeper explanation, which is out of scope of this paper.

To sum up, we can say, that ES5 supports first-class functions, but it lacks other properties of functional programming. Some of them can be simulated indirectly or using libraries. Functional style is not idiomatic in standard JavaScript, but a number of libraries and projects leveraging them seems to be rising.

4.2 ES6

ES6 is the new (2015) JavaScript standard and it is backward compatible. It brings several improvements, which are explained e.g. in [31]. Let us discuss the changes related to the functional principles here.

The first improvement is a more elegant syntax for anonymous (Lambda) functions: the //arrow functions//. The code:

```
evens.map(function (v) { return v + 1 })
```

can be shorten to

```
evens.map(v => v + 1)
```

So we may say that ES6 syntactically supports (P1) better than ES5. Also, by using arrow functions, one can achieve a more expressive closure syntax (P4). Also, ES6 changes scoping of `this` from function scope to lexical scope. This change allows a direct closure code – we do not have to save `this` reference as in ES5. Instead of

```
var self = this
this.nums.forEach(function (v) {
    if (v % 5 === 0)
        self.fives.push(v)
})
```

it is now possible to write [32]:

```
this.nums.forEach((v) => {
    if (v % 5 === 0)
        this.fives.push(v)
})
```

ES6 also improves the support for immutability. In the language, there are now two new keywords for creating variables. Keyword `let` creates mutable variable but scoped lexical (instead of the functional scoped `var` keyword). Using `const` one can create a constant. The ES6 `const` keyword is used to declare read-only variables, i.e. the variables whose value cannot be reassigned [31]. So (P3)

is supported, but not required. Immutable variable means that nothing else can be assigned to this variable, but properties of this object still can be changed.

ES6 also adds *collections* and new functions for immutable operations: `find()`, `map()`, `reduce()` do not change the original collection. This strongly supports referential transparent code (P3).

A support for *tail call optimization* has been added, which facilitates the usability of recursion (P5). A tail position call must either release any transient internal resources associated with the currently executing function execution context before invoking the target function or reuse those resources in support of the target function [22].

ES6 supports *generators*, which are essentially lazy collections. One can define an iterable sequence with generator function and the control flow can be paused and resumed, in order to produce a sequence of values (either finite or infinite). But this is not enough to support laziness (P6). This provides only a type of lazy collection, but there is still missing lazy function invocation or lazy values.

Currying (P7) is still not part of the language, the situation remains the same as with ES5: it is achievable using libraries.

As for the additional principles, ES6 introduced the *destructuring assignment*:

```
var list = [ 1, 2, 3 ]
var [ a, , b ] = list
[ b, a ] = [ a, b ]
```

It is essentially (P8): pattern matching, but just in a limited fashion, as it can not be used to create polymorphic functions based on destructuring.

The model of (P9): polymorphism has not changed in ES6.

5 Conclusions

It may be an interesting observation that JavaScript in spite of its C-like syntax offers many fundamental functional programming principles in its heart and a lot can be achieved by libraries. Programmers may thus leverage a good portion of functional power and elegance in this popular language. ES6 is obviously a step towards better functional principles support. JavaScript is one of the languages, which is getting closer to purely functional languages. The following table summarizes concepts and their support in languages.

Property	ES5	ES6
First-class functions (P1)	+	++
Referential transparency (P2)	–	-
The immutability of variables and values (P3)	-*	+
Closures (P4)	+	++
Recursion (P5)	+	++
Lazy evaluation (P6)	–	-
Currying (P7)	-*	-*
Pattern matching (P8)	–	-
Polymorphism (P9)	+	+

* can be reached using libraries.

As for the future work, the presented conceptual framework may be used to analyse other programming languages from the functional programming perspective.

References

1. "List of programming languages," Nov. 2016, page Version ID: 750948731. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_programming_languages&oldid=750948731
2. S. Kedar, *Programming Paradigms And Methodology*. Technical Publications, Jan. 2008, google-Books-ID: gvm9TPE96t4C.
3. "Clojure." [Online]. Available: http://clojure.org/
4. R. Warburton, *Java 8 Lambdas: Pragmatic Functional Programming*. "O'Reilly Media, Inc.", Mar. 2014, google-Books-ID: qKUdAwAAQBAJ.
5. *Computer Science*. PediaPress, google-Books-ID: Yte2cXVES9EC.
6. G. Cousineau and M. Mauny, *The Functional Approach to Programming*. Cambridge University Press, Oct. 1998, google-Books-ID: vccmAAAAQBAJ.
7. J. Hughes, "Why Functional Programming Matters," *The Computer Journal*, vol. 32, no. 2, pp. 98–107, Jan. 1989.
8. L. VanderHart and S. Sierra, *Practical Clojure*, 1st ed. Apress, Jun. 2010, 00000.
9. R. Bird, *Thinking Functionally with Haskell*. Cambridge University Press, Oct. 2014, google-Books-ID: B4RxBAAAQBAJ.
10. B. O'Sullivan, J. Goerzen, and D. B. Stewart, *Real World Haskell: Code You Can Believe In*. "O'Reilly Media, Inc.", Nov. 2008, google-Books-ID: nh0okI1a1sQC.
11. J.-P. Jouannaud, *Functional Programming Languages and Computer Architecture: Proceedings, Nancy, France, September 16-19, 1985*. Springer Science & Business Media, Sep. 1985.
12. R. Pickering and K. Eason, *Beginning F# 4.0*. Apress, May 2016, google-Books-ID: puQgDAAAQBAJ.
13. K. Koval, *Swift High Performance*. Packt Publishing Ltd, Nov. 2015, google-Books-ID: VfioCwAAQBAJ.

14. K. Ballou, *Learning Elixir*. Packt Publishing Ltd, Jan. 2016, google-Books-ID: ogUcDAAAQBAJ.
15. A. Alexander, *Scala Cookbook: Recipes for Object-Oriented and Functional Programming*. "O'Reilly Media, Inc.", Aug. 2013, google-Books-ID: BSo2AAAAQBAJ.
16. L. Borges, *Clojure Reactive Programming*. Packt Publishing Ltd, Mar. 2015, google-Books-ID: 1tePBwAAQBAJ.
17. F. Geller, R. Hirschfeld, and G. Bracha, *Pattern Matching for an Object-oriented and Dynamically Typed Programming Language*. Universittsverlag Potsdam, 2010, google-Books-ID: 2gM_93yaz_kC.
18. Z. Hu, J. Hughes, and M. Wang, "How functional programming mattered," *National Science Review*, vol. 2, no. 3, pp. 349–370, Sep. 2015, 00000.
19. P. W. Luca Cardelli, "On Understanding Types, Data Abstraction, and Polymorphism," Dec. 1985. [Online]. Available: http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf
20. D. Crockford, *JavaScript: The Good Parts: The Good Parts*. "O'Reilly Media, Inc.", May 2008, google-Books-ID: PXa2bby0oQ0C.
21. "ECMAScript Language Specification," Dec. 2009. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262%205th%20edition%20December%202009.pdf
22. "ECMAScript 2016 Language Specification," Jun. 2016. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
23. "ECMAScript 6 compatibility table," Nov. 2016. [Online]. Available: http://kangax.github.io/compat-table/es6/
24. "Lodash." [Online]. Available: https://lodash.com/
25. "Underscore.js." [Online]. Available: http://underscorejs.org/
26. "Immutable.js." [Online]. Available: https://facebook.github.io/immutable-js/
27. "amdjs/amdjs-api." [Online]. Available: https://github.com/amdjs/amdjs-api
28. "A Beginner's Guide to Currying in Functional JavaScript," Oct. 2015. [Online]. Available: https://www.sitepoint.com/currying-in-functional-javascript/
29. "Ramda Documentation." [Online]. Available: http://ramdajs.com/
30. M. E. Daggett, *Expert JavaScript*. Apress, Nov. 2013, google-Books-ID: HpoQAwAAQBAJ.
31. N. Prusty, *Learning ECMAScript 6*. Packt Publishing Ltd, Aug. 2015, google-Books-ID: 9O13CgAAQBAJ.
32. "ECMAScript 6: New Features: Overview and Comparison." [Online]. Available: http://es6-features.org/#Lexicalthis

Chapter 20

OpenCASE — A Tool for Ontology-Centred Conceptual Modelling

[222] Pergl, R.; Tůma, J. OpenCASE - A tool for ontology-centred conceptual modelling. Lecture Notes in Business Information Processing, volume 112 LNBIP, 2012: pp. 511–518.

OpenCASE– A Tool for Ontology-Centred Conceptual Modelling

Robert Pergl and Jakub Tůma

Department of Information Engineering,
Faculty of Economics and Management,
Czech University of Life Sciences,
Prague, Czech Republic
pergl@pef.czu.cz, jtuma@pef.czu.cz

Abstract. OpenCASE, an original CASE tool supporting conceptual modelling is presented in this paper. The CASE tool has been developed during the research focused on the ontology-centred conceptual modelling. It provides a strong emphasis on terms and their relations while supporting standard notations (now BORM, other notations are planned). The tool has an open plug-in-based architecture founded on the Eclipse platform, which makes the tool modular and extensible. The knowledge base of the models may be accessed via an API and thus used to implement verifications, various calculations (statistics), to transform models to outputs (reports) and to make inner transformations (e.g. normalisation). The architecture of the tool is briefly mentioned as well.

Key words: CASE Tool, Eclipse platform, conceptual modelling, ontological analysis, BORM method

1 Introduction

This contribution addresses the discussion of the importance of diligent ontological analysis during the enterprise IS modelling presented in [13], where the author explains the importance of ensuring the *consistency* between various models (and inside each model) and concludes (besides others) the need of a "quality CASE tool support".

In this paper, we would like to present our advancements in designing and implementing a CASE tool to support ontology-centred modelling: OpenCASE. OpenCASE [12] is a CASE tool designed to support the research in the field of conceptual modelling and ontologies. It is built upon the Eclipse framework [8] and it utilizes many of its advanced possibilities (see section 4). Right now, we have implemented the BORM method's Business Architecture Diagrams and Object Relation Diagrams ([7],[1], [10]) as a proof-of-concept of ontology-centred modelling and OpenCASE's philosophy and design[1].

[1] Apologies for the readers: due to lack of space in this short paper, we do not provide a BORM introduction here.

2 Goal and Methodology

The goal of the contribution is to present `OpenCASE`, an original ontology-centred CASE tool. We present the philosophy behind the tool, its practical utilisation and its architecture.

3 Ontology-Centred Modelling

3.1 Entities vs. Elements

To deal with ontology generally means to deal with *terms* and their *relations*[2]. Conceptual modelling using the ontology-centred approach thus needs to fully support *tracking of distinct terms and their relations* throughout the models. This practically means a transition from *visually-centred* to the *ontology-centred* CASE tools architecture. We may find notions of this approach in some CASE and Meta-CASE tool like Craft.CASE [3], MetaEdit+ [11] and others, however we were focused on total concept purity in separating the Domain Layer and the Domain Model Layer while maintaining the relation between the elements (Figure 1[3]).

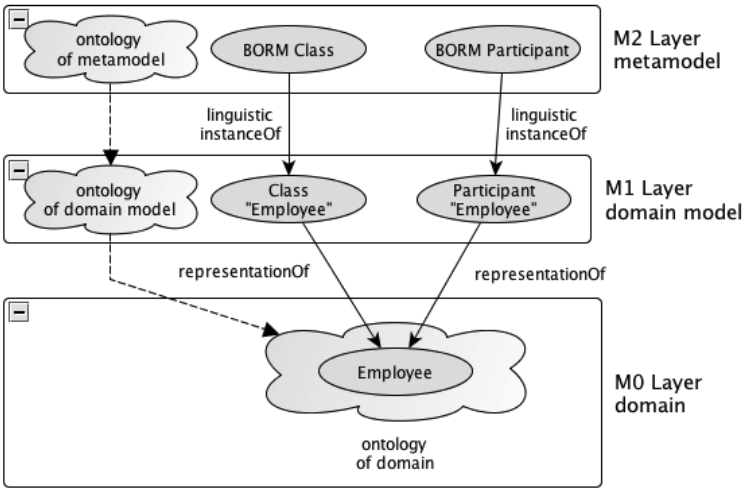


Fig. 1. Layered architecture of ontologies (taken from [13])

We implemented the concept of *ontologically equivalent elements* introduced in[13]:

[2] A more thorough introduction to ontologies and their relation to conceptual modelling and the BORM methodology may be found in our original paper [13].

[3] The layered architecture of ontologies has already been published in its rudimentary form numerus times over the past 30 years, see e.g. [4].

Definition 1. *We say that the model element x is* ***ontologically equivalent*** *to the model element y if and only if there exist relations $representationOf(x,t)$ and $representationOf(y,t)$, where t is an element of the CMoD*[4]*.*

We implemented this concept by strictly discerning the domain *entities* and model's *elements*. An "entity" represents a domain object in the M0 layer in Figure 1, while the "element" is an object in the M1 layer, being a particular instance of the M2 layer element. Practically, let us suppose we deal with a `Customer` entity being modelled by a Participant `Customer` graphically represented according to the BORM methodology as a rectangle with a solid border and pale blue filling, which is an instance of the `Participant` concept.

There is a relation 1:N between entities and elements: each entity (layer M0) may be represented by several elements (layer M1): a Customer may play its role in several diagrams, thus being represented by a visual element in each, while being the same entity's *representationOf*. There is a screenshot in Figure 2 showing the `OpenCASE`'s support for tracking entities with respect to elements. The left panel shows all the entities. If we unfold an entity, we see all its elements. In Figure 2 there are 3 participant elements that represent entity `Customer`[5]. When we click an element, its full path is revealed in the status line. Double-clicking an element takes us to the appropriate diagram and selects the element.

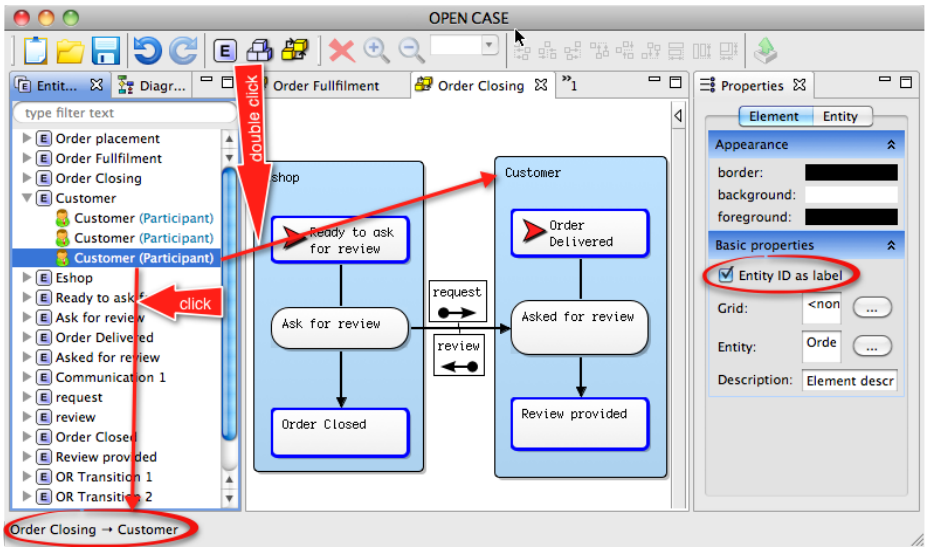


Fig. 2. Entities and Elements in `OpenCASE`

Tracking the entities is a concept that enables us to:

[4] CMoD = Concept Map of Domain ... a graph of domain terms and their relations.
[5] Generally, there may be various element types representing one entity, e.g. there may be as well a data class `Customer` that would describe customer's attributes.

Ensure elements' consistency – Elements usually take the name of their entity, so renaming an entity automatically renames all the attached elements. Thus if we have an element `Customer` participating in several diagrams and we change its entity's name to `Client`, all the attached elements get automatically renamed. In case we do not want an element to automatically take its entity's name – this may be the situation in multilingual diagrams or languages that use inflection – we may disable the implication "entity name → element name" (option `Entity ID as label`, red ellipse on the right in Figure 2). We need to rename the element by hand, however with the comfort of having the list of all elements that are *representationOf* the entity.

Facilitate impact analysis – When some change occurs to a domain entity, we may easily track the impact to the model, i.e. the elements in the model that may need attention due to the change.

Use the model knowledge base – Elements represent some information we have about the entity – its roles and relations in the domain. We may design and run reports, statistics, optimizations and reasoning based on this knowledge. `OpenCASE` provides a full API to this model knowledge base – see subsection 3.4.

3.2 Business Properties

Another concept of ontology-centred modelling implemented in `OpenCASE` are *business properties*. Inspired by the success of this concept in the Craft.CASE tool [3], we implemented a sort of meta-modelling layer enabling to specify custom domain (business) properties. Compared to Craft.CASE we did not limit business properties just to classes of elements (Participant, State, Activity, ...), but we made possible to attach a business property to an element (thus having just one instance) or to an element class (thus having several instances) – Figure 3. Because diagrams are elements as well, they may have business properties orthogonally assigned, too (the author, version, etc.).

3.3 Internal Knowledge Base

The tool is database-centred, i.e. it maintains an internal knowledge base containing of functions, scenarios, diagrams, entities, elements together with their graphical properties. Internally, they form nodes of a graph structure and their relations are represented as edges, thus various graph algorithms may be applied on the knowledge base [17]. Graph traversals and graph transformations are probably the most useful and enable to implement operations like

- *Listings*, like all input/output flows from/to a participant.
- *Calculation of metrics* (like numbers of states and activities in participants) that may be used for *complexity estimations* [16], [15].
- *Calculation of statistics*, e.g. about dataflows and communications (which participants communicate the most/least, above/below average, etc.).

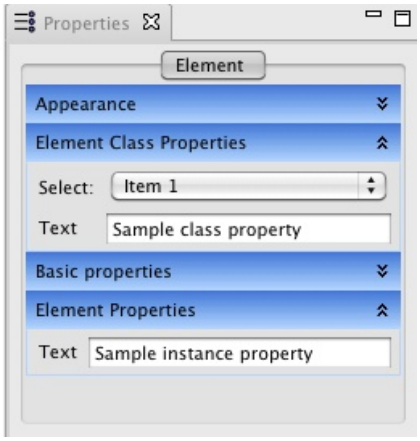


Fig. 3. Element Properties and Element Class Properties in `OpenCASE`

– *Semantics checks*: there is a starting state in every participant, at least one final state[6], ...
– Conceptual normalisations [9].
– Any further custom reporting / calculations / processing

3.4 Model API

We see diagrams as a convenient way how to specify the model and visualize the model to a business user, however the true power lies in its underlying knowledge base. We designed `OpenCASE` to transparently reveal its API (Application Programming Interface) of the model's structure. Using this API, a programmer may iterate through the model's elements and entities, make verifications, perform various calculations (statistics), transform them to some sort of output (reports) and make inner transformations (e.g. normalisation).

The API is self-documented in the form of UML Class diagrams thanks to the Ecore framework (see section 4). An example of the OR diagram metamodel is in Figure 4.

4 `OpenCASE` Implementation

`OpenCASE` is implemented entirely in the Java programming language utilizing the Eclipse Platform and various modelling frameworks from the Eclipse Modeling Project (EMP). The Eclipse Rich Client Platform (RCP) offers a very powerful foundation since it provides an extensible component system and a platform for creating complex applications with rich user interfaces.

[6] According to the BORM method, there may be exceptions to these rules, see [7] for more details.

Due to lack of space in this short paper, we do not provide an overview of the RCP platform. The reader may read about it on the Internet or in the literature – we highly recommend [2], [5], [8], [14].

The core of the `OpenCASE` project is the `OpenCASE` written as an RCP application. There are 10 essential plug-ins constituting the core of the application called `OpenCASE` Workbench. The workbench is just the user interface without any diagram editing capabilities. Diagram manipulation is performed by the remaining plug-ins.

Each feature has a core plug-in having an ID with no suffix, e.g., `org.opencase.diagrams`. Such a plug-in is almost entirely generated from an Ecore model and it implements the basic behaviour of the modelled domain. An example of Ecore model is in Figure 4.

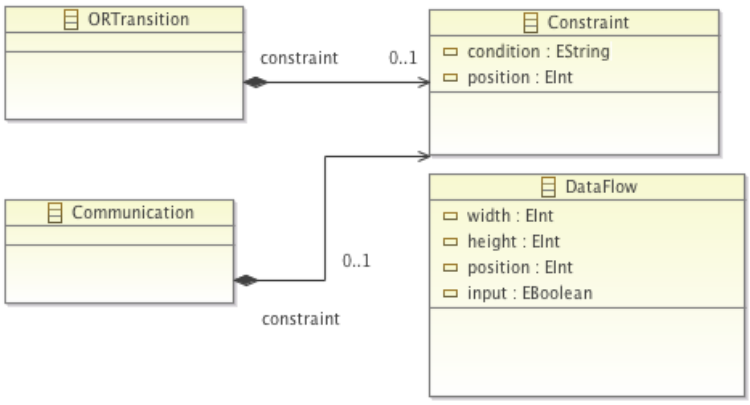


Fig. 4. A part of Ecore model of BORM's Object Relationship Diagram

Currently, there are several plug-ins being developed, compiled and deployed in separation from the core of `OpenCASE`. Thanks to Eclipse plug-in system such components can be installed right into the running `OpenCASE` from an archive or internet update site.

5 Summary, Conclusions and Future Work

`OpenCASE` is an attempt to bring the ontology-centred modelling into everyday life and profit from research achievements while at the same time to provide an open platform for further research. That provided a challenge to implement theoretical results into suitable software implementation and to build a user-friendly tool with features like keyboard shortcuts, complete undo, aligning and distribution of graphical elements, batch operations, etc.

We put a high focus to implement the whole ontology chain, i.e.

1. Input – *How to input the terms and their relations in synergy with a concrete notation.* We addressed this issue by separating the identity (entity) from its representation (element) – subsection 3.1
2. Processing – *How to access the ontology and manipulate it by transformations and various algorithms* (verifications, normalizations, optimizations, etc.). We built an application programming interface (API) to access the model's knowledge base. The API architecture is documented by UML (Ecore) diagrams[7] – subsection 3.4.
3. Output – *How to export the ontological knowledge contained in the model.* Exporters plug-ins handle this task. Exporter plug-ins are implemented as Eclipse plug-ins and they may be implemented to perform an export to various human-readable formats (TXT, HTML, LaTeX, ODT, PDF, ...) or formats suitable for machine processing (CSV, XML, JSON, OWL, ...), or it may perform the export directly into relational database or reveal the knowledge base as a service (SOAP, REST).

At the time of writing this contribution, the modelling core is completely implemented, being further fine-tuned and improved. As for the plug-ins, several output plug-ins are developed (TXT, HTML, LaTeX). We are also working on implementing models simulations and support for optimizations. A huge step toward the holistic ontology-centred conceptual modelling will be implementing other types of diagrams, especially data-structure diagrams (UML Class Diagrams and OntoUML, [6]) and providing a means to make ontologic relations to the process diagrams.

Acknowledgements. ***

References

1. Brozek J., Merunka V., Merunkova I.: Organization Modeling and Simulation Using BORM Approach. In: Lecture Notes in Business Information Processing, vol. 63, pp.27-40 (2010)
2. Clayberg E., Rubel D.: Eclipse Plug-ins. Addison-Wesley Professional (2008)
3. Craft.CASE Tool, `http://www.craftcase.com`
4. Gasevic, D., Djuric, D., Devedzic, V.: *Model Driven Engineering and Ontology Development (2nd Edition)*, Springer (2009).
5. Gronback R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional (2009)
6. Guizzardi G.: Ontological Foundations for Structural Conceptual Models. Telematica Instituut Fundamental Research Series No. 15 (2005)
7. Knott, R.P., Merunka, V., Polák, J.: The BORM methodology: a third-generation fully object-oriented methodology. In: Knowledge-Based Systems, vol. 16, no. 2, pp. 77-89 (2003)

[7] Actually, as was explained, the situation is opposite: the internal structures are *generated* from the Ecore diagrams, which makes a powerful mechanism to maintain specification-implementation consistency. Nevertheless, this is irrelevant for the API's user.

8. McAffer J., Lemieux J.-M., Aniszczyk Ch.: Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications. Addison-Wesley Professional (2005)
9. Molhanec, M.: Some Reasoning Behind Conceptual Normalisation. In: Information Systems Development Information Systems Development, pp. 517-525, Springer Science+Business Media, Berlin (2011)
10. Molhanec, M., Merunka, V.: BORM: Agile Modelling for Business Intelligence In: Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications. Hershey, Pennsylvania: IGI Global, pp. 120-130 (2011)
11. MetaEdit+, `http://www.metacase.com/cases/borm.html`
12. `OpenCASE` Tool, `http://www.opencase.net` (in construction)
13. Pergl R.: Supporting Enterprise IS Modelling using Ontological Analysis. In: Lecture Notes in Business Information Processing, vol. 88, no. 1, pp. 130-144, Springer, Heidelberg (2011)
14. Steinberg D., Budinsky F., Paternostro M.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2008)
15. Struska Z., Merunka V.: BORM points – New concept proposal of complexity estimation method. In: ICEIS 2007: PROCEEDINGS OF THE NINTH INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS: INFORMATION SYSTEMS ANALYSIS AND SPECIFICATION, 9th International Conference on Enterprise Information Systems (ICEIS 2007), Funchal, PORTUGAL, JUN 12-16, 2007, pp. 580-586 (2007)
16. Struska Z., Pergl R.: BORM-points: Introduction and Results of Practical Testing. In: ENTERPRISE INFORMATION SYSTEMS, Lecture Notes in Business Information Processing, vol. 24, pp. 590-599 (2009)
17. Valiente G.: Algorithms on Trees and Graphs. Springer (2010)

Chapter 21

BORM-II and UML as Accessibility Process in Knowledge and Business Modelling

[223] Merunka, V.; Pergl, R.; Tůma, J. BORM-II and UML as Accessibility Process in Knowledge and Business Modelling. In New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering, number 312 in Lecture Notes in Electrical Engineering, Springer International Publishing, 2015, ISBN 978-3-319-06763-6, pp. 1–6.

BORM-II and UML as accessibility process in knowledge and business modelling

Vojtěch Merunka, Robert Pergl, Jakub Tůma

***Abstract*—This paper presents two systems and knowledge modelling techniques that may be used as a tool to coordinate the communication between researchers and users from the agriculture problem domain. The paper is focused on th eusage of a general approach UML (Unified Modelling Language) and an innovative approach BORM-II (Bussiness Object Relation Modelling, second generation) as communication standards within research projects. The first part of this paper describes the framework, laying out the main aspects of both notations, metamodel and theoretical background as well as their advantages and disadvantages. The paper analyses practical examples from agriculture, rural and organization modelling domains. These innovation processes in both approaches are aplied on the same business process description and evaluetes the impact on researchers and users of research. The main part is focused on the transformation model to model based on BORM-II. The transformation is in line with UML and SBVR (Semantics of Business Vocabulary and Rules) standards from OMG (Object Management Group). My predecessor worked on model transformation BORM to UML. This work follows Petr Šplíchals work and goes further. This transformation will be composed into a modelling tool and willl be based on approach HOT (High Order Transformation). The objective of this research is to achieve documentation output like SBVR, and bridge the gap between business people (users) and designers (researchers) of information systems (IS). The paper concludes that the gap between IS designers (software engineers) and domain experts can be bridged by automated transformation of previously mentioned models. The main goal is to achieve a documentation output similar to SBVR, and ICT Accesibility for business people.**

***Index Terms*—Business Process and Knowledge Modelling, ICT Accessibility, Unified Modelling Language, Business Object Relation Modelling, Model Transformation, Tool, Semantics of Business Vocabulary and Rules**

Manuscript received November 11, 2013. This work was supported in part by internal grant agency ČZU v Praze IGA number 20121059, Metody automatizovaných transformací modelů v informačních systémech.

V. Merunka, J. Tůma are with Department of Information Engineering, FEM, Czech University of Life Sciences Prague, Kamycka 129, 165 21 Prague 6 Suchdol (e-mail: merunka, jtuma@pef.czu.cz).

R. Pergl is with Department of Information Engineering, FEM, Czech University of Life Sciences Prague, Kamycka 129, 165 21 Prague 6 Suchdol (e-mail: pergl@pef.czu.cz).

I. Introduction

THIS paper presents two system and knowledge modelling techniques that may be used as a tool to coordinate the communication between researchers and users from the agriculture problem domain. The paper is focused on the use of a general approach UML (Unified Modelling Language) [2] and an innovative approach BORM-II (Business Object Relation Modelling, second generation) as communication standards within research projects.

A. BORM

Business Object Relationship Modelling [6], [11] is a object-oriented software engineering methodology, which has proven to be very effective in the development of business information and knowledge systems. Its effectiveness is achieved by unified and simple method for presenting all aspects of relevant model. The BORM methodology makes extensive use of business process modelling [5]. BORM was designed as a method covering all phases of software development. BORM focuses mainly on the first phases of the project also known as business analysis. BORM uses only limited, easily comprehensible group of concepts for every life-cycle phase. This makes it easier to understand even for the first-time users with almost no knowledge of business analysis.

Another fact that makes the BORM methodology more expressive is that it doesnt need the division to static and dynamic views of the model and therefore does not bring a need of creation of different diagrams with a different view points. BORM introduces the following types of diagrams:

- Business architecture diagram
- Object relationship diagram
- Class diagram

BORM represents every concept with the same symbols in the data structure, the communication or other diagrams. For visual presentation of the information BORM uses simple diagrams that contain only a necessary number of concepts and symbols. These concepts and symbols cover most of the needs for the initial description of the model and its processes. The following symbols belong to the symbols used in the initial description:

- Participant an object representing the stakeholder involved in one of the modelled processes, which is recognised during the analysis.
- State sequential changes of the participants in time are described by these states.
- Association data-orientated relation between the participants.
- Activity represents an atomic step of the behaviour of the object recognised during the analysis.
- Communication represents the data flow and dependencies between the activities. Bidirectional data flow may be present during the communication.
- Transition connects the state-activity-state and represents changes of the states through activities.
- Condition expresses constraint that holds for the communication or activity, [6].

B. UML

The Unified Modelling Language (UML) is a standardised notation for specifying object-oriented software systems (Booch et al, 1999), (OMG, 1999), (Rumbaugh et al, 1999). A UML model is a set of diagrams describing and documenting the structure, behaviour and usage of a software system. UML is used to model all kinds of software systems, including concurrent and embedded systems. There are commercial modelling tools available on the market to help the designer in creating the UML models and these tools can also generate program code from some diagrams of the model. UML is an expressive and rich language, their models must still be verified, since a model may contain unexpected behaviours from the designer.

II. MATERIALS AND METHODS

A. Goal

The goal of this contribution is to present an approach for flexible modelling of business processes both at the management and operations level. The approach consists of combining a suitable modelling method and developing an original software tool to support it, as well as to perform automated model transformations.

The goal presented another view-point of this research is achieving documentation output like SBVR, [9], and bridge the gap between business people (users) and designers (researchers) of information systems (IS) [15].

B. Methodology

- 1. First we set requirements for a suitable management-level business process model and operation-level business process model (BPM).
- 2. We describe how the selected modelling method and the OpenCASE support these requirements.
- 3. We present a case study to illustrate the results.

Requirements for Management-Level and Operation-Level BPM For our purpose, let's define the management level is focused on the process orchestration, specifically:

- 1. terminology,
- 2. the logic of processes,
- 3. the relations of processes (transition, decomposition, ,
- 4. communications between participants,
- 5. optimisation of the overall process.

For the management level, the language of the model needs to support the mentioned aspects. This is why usually a combination of graphical and textual language is used.

The management level BPM specifies terms and their relations that are consequently manipulated in different ways:

- They need to be verified for correctness.
- They need to be communicated.
- They are used for reasoning.
- Various reports and statistics need to be calculated.
- They are often changed (they evolve).

This is why the management-level BPM needs to be sort of knowledge base, not just a set of diagrams (graphical objects). By the operational level here, we mean concrete process participants (staff, systems) performing the specified processes. For this level, we specify the following requirements on the operation model:

- The language of the model is close to the language of the participant.
- The model is accurate.
- The model contains just necessary details to perform the operations.
- The model is up to date and consistent with the management-level model.

As systems participants provide quite a different category (being software and thus computer science and software engineering methods apply here), we will consider just human participants (staff) here. Staff at the operation level are not supposed to be interested in the big picture they just need accurate instructions for performing their tasks, i.e. the management needs to answer their questions:

- What are the steps I should follow to successfully complete a task?
- How should I make decisions and select correct approach?
- What are the inputs that I will get? From whom, how and when?
- What are the outputs that I should produce? To whom shall I handle them, how and when?

For operation levels, usually textual operation manuals are used, as operation-level staff is not supposed to prefer abstract notations.

III. RESULTS AND DISCUSSION

The case study demonstrates the transformation from the management-level business process model into the operation-level business process model. As we specified in requirements, the operation-level model should be textual and tailored for each participant. This is where we utilize the OpenCASE

the knowledge base and API and generate HTML page for each participant. HTML documentation output is like SBVR, [9]. This is achieved by selecting the Project - Generate Report menu item in OpenCASE. Generation is based on publication [13] and this transformation is composed of a modelling tool and based on approach HOT (High Order Trasformation) published by [3].

The case study deals with the process of suspicious identification method (MPI) used in criminalistic. Just a part of the process is presented here due to space limitations. The case study is focused on collecting suspicious samples and their analysis. The process covers a collection of suspicious samples. The case study was written with cooperation of the Faculty of Agrobiology, Food and Natural Resources, namely we would like to thank Ing. Petr Vlasak from Canine Behavior Research Center. Here we will use just a simplified version to demonstrate the concepts presented in the paper.

The MPI process is carried out by cooperation between four participants: Regional institution Distribution, Territorial criminal technician, Regional institution Analysis and Inspector. The whole case study is shown in figure 1 and in detail (Figure 6-9).

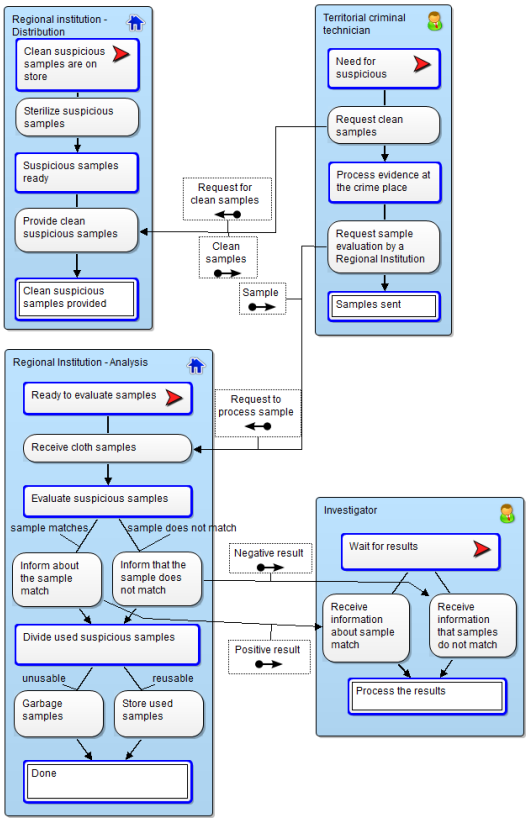


Fig. 1. Case study management process model in BORM: Partial Phase of MPI.

Regional institution - Distribution
Organization

§1	a) Clean suspicious samples are on store
§2	a) Sterilize suspicious samples
§3	a) Suspicious samples ready
§4	If "Request for clean samples" recieved from "Territorial criminal technician": a) Provide clean suspicious samples *Send "Clean samples" to "Territorial criminal technician" as response to "Request for clean samples".*
§5	a) Clean suspicious samples provided

Fig. 2. Operation model (manual) for participant Regional Institution Distribution.

Territorial criminal technician
Person

§1	a) Need for suspicious samples
§2	a) Request clean samples *Send "Request for clean samples" to "Regional institution - Distribution" and receive "Clean samples" in response.*
§3	a) Process evidence at the crime place
§4	a) Request sample evaluation by a Regional Institution *Send "Request to process sample", "Sample" to "Regional Institution - Analysis".*
§5	a) Samples sent

Fig. 3. Operation model (manual) for participant Territorial Criminal Technician.

Regional Institution - Analysis
Organization

§1	a) Ready to evaluate samples	
§2	If "Request to process sample" and "Sample" recieved from "Territorial criminal technician": a) Receive cloth samples	
§3	a) Evaluate suspicious samples Go to §4a or §4b according to entrance conditions.	
§4	If sample matches a) Inform about the sample match *Send "Positive result" to "Investigator".*	If sample does not match b) Inform that the sample does not match *Send "Negative result" to "Investigator".* Go to §5a
§5	a) Divide used suspicious samples Go to §6a or §6b according to entrance conditions.	
§6	If unusable a) Garbage samples	If reusable b) Store used samples Go to §7a
§7	a) Done	

Fig. 4. Operation model (manual) for participant Regional Institution Analysis.

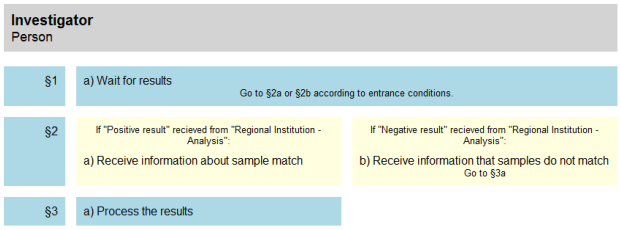


Fig. 5. Operation model (manual) for participant Investigator.

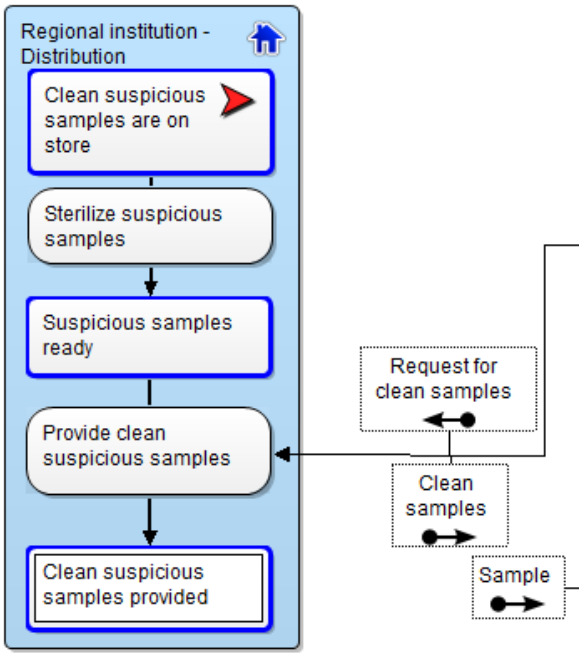


Fig. 6. Case study management process model in BORM: Partial Phase of MPI, detail of distribution.

The exporter then traverses through the inner diagram structure. The HTML operation manuals are generated for each participant (Figure 2-5).

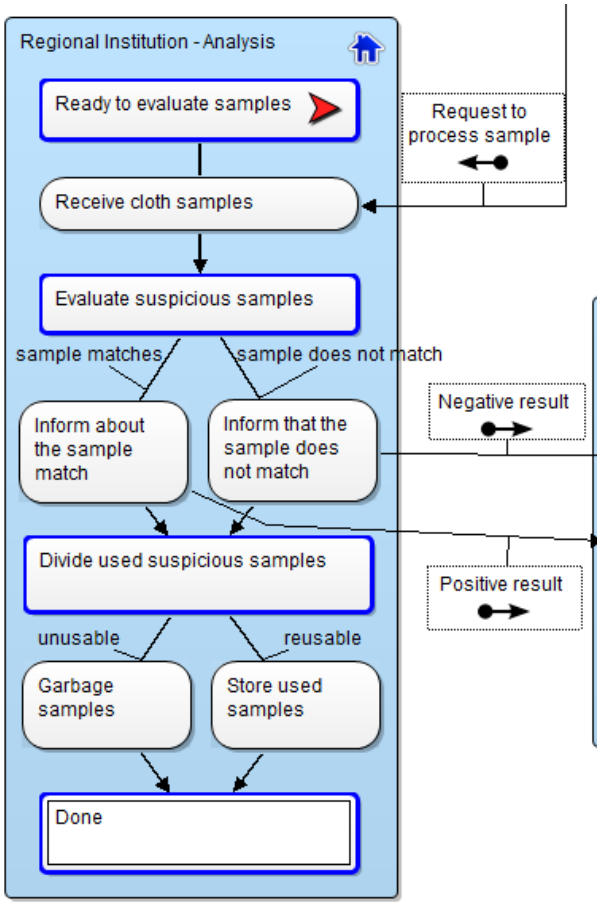


Fig. 7. Case study management process model in BORM: Partial Phase of MPI, detail of analysis.

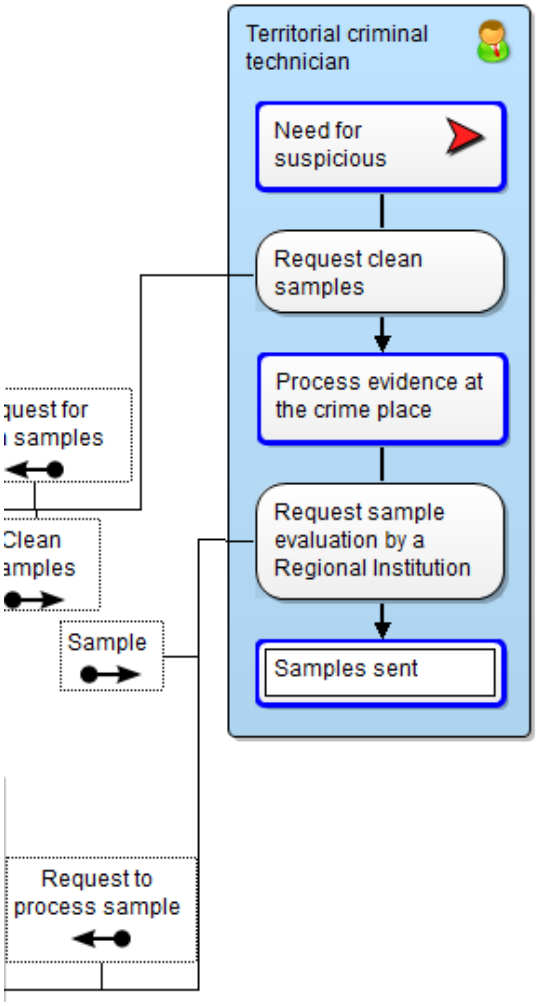


Fig. 8. Case study management process model in BORM: Partial Phase of MPI, detail of technician.

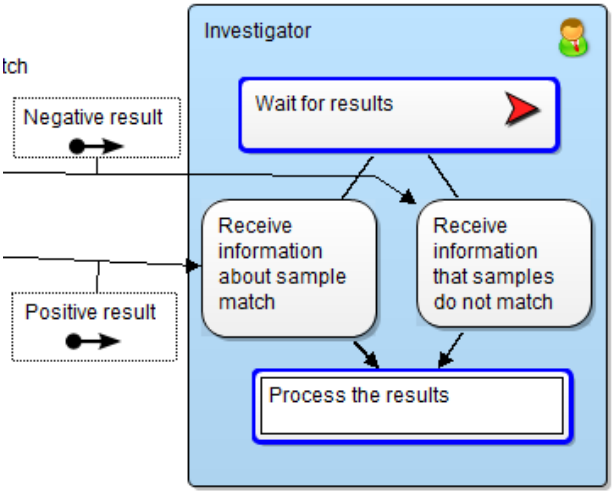


Fig. 9. Case study management process model in BORM: Partial Phase of MPI, detail of investigator.

IV. Conclusion

In this paper we presented our solution that supports business process engineering. It is a combination of a suitable method and notation (BORM) supported by a software tool (OpenCASE).

The key aspect of the solution is that the modelled process is not just a diagram, but a whole knowledge base that may be used in operations, reporting, decision making and other areas. We presented one of its possibilities: automatic generation of operations manuals. Other possibilities include:

- Listings, like all input/output flows from/to a participant.
- Calculation of metrics (like numbers of states and activities in participants) that may be used for complexity estimations [16], [14].
- Calculation of statistics, e.g. about data flows and communications (which participants communicate the most/least, above/below average, etc.).
- Semantics checks: there is a starting state in every participant, at least one final state3,
- Conceptual normalization [7].
- Any further custom reporting / calculations / processing

These areas provide many topics both for practice and research. The OpenCASE is implemented using open architecture based on Eclipse plugins, which makes it easily extensible and thus provides a platform for further studies. Apart from this, it is already a stable tool for effective drawing of BORM and their management.

Future work statistical research of developed methods in practice is planned.

Acknowledgment

Many thanks to my supervisor Vojtěch Merunka and to

the team group leader Robert Pergl for a lot hints and mental support. This paper documents summarise previous and future developing process of disertation thesis. This article was realized with support of internal grant agency ČZU v Praze, IGA project number 20121059, Metody automatizovaných Transformací modelů v informačních systémech. I would like to thank proof readers Sarah Marlena, Angela Bilbilovska and Juraj Kardoš.

REFERENCES

[1] H. Kopka and P.W. Daly, A Guide to LATEX, 3rd edition Harlow, England: Addison-Wesley, 1999.
[2] Booch, G., Rumbaugh, J., and Jacobson, I. . (1999) The Unified Modelling Language User Guide. Addison-Wesley.
[3] Brambilla, M., Fraternali, P. and Tisi, M., A Transformation Framework to Bridge Domain Specific Languages to MDA. MoDELS Workshops 2008: pp. 167-180.
[4] Gronback, R., C. (2009) Eclipse Modelling Project: A Domain Specific Language (DSL) Toolkit. Addison-Wesley Professional.
[5] R. Knott, V. Merunka, J. Polák, Process Modelling for Object Oriented Analysis, In Proceedings of Fourth International Conference on Requirements Engineering, IEEE ACM, Chicago 2000.
[6] R. Knott, V. Merunka, J. Polák, (2006) Chapter 15: The BORM Method: A Third Generation Object-Oriented Methodology. In: Liu, L., Roussey B. (eds) Management of the Object-Oriented Development Process. Pp. 337-360. IGI Publishing, [s.1.], ISBN 9781591406044.
[7] M. Molhanec, (2011) Some Reasoning Behind Conceptual Normalisation. In: Inforamtion Systems Development Information Systems Development, Springer Science+Business Media, Berlin, pp. 517-525.
[8] OMG Unified Language Specification (draft). Version 1.3 alpha R5, March 1999, available at http://www.rational.com/uml/resources/documentation/media/OMG-UML-1_3-Alpha5-PDF.zip.
[9] OMG: Semantics of Business Vocabulary and Rules (SBVR) Specification, v 1.0 (formal/08-01-02), 2008.
[10] OMG: UML 2.1.1 Superstructure Specification, OMG Adopted Specification (formal/07-02-03), 2007.
[11] J. Polák, V. Merunka, A. Carda (2003) Umění systémového návrhu: objektově orientovaná tvorba informačních system pomocí původní metody BORM. Grada, Prague, ISBN 80-247-0424-2..
[12] J. Rumbaugh, J. Jacobson, G. Booch, (1999) The Unified Modelling Language Reference Manual. Addison-Wesley.
[13] P. Splichal, R. Pergl, M. Pícka (2011) "BORM Model Transformation", P. Šplíchal (2011) Model Transformation, Agrarian perspectives proceeding of the 20th International Scientific Conference, Prague, pp. 423-430..
[14] Z. Struska, V. Merunka (2007) BORM points New concept proposal of complexity estimation method. In: ICEIS 2007: Proceedings of the ninth international conference on enterprise information systems: information systems analysis and specification, 9th International Conference on Enterprise Information Systems (ICEIS 2007), Funchal, Portugal, JUN 12-16, 2007, pp. 580-586.
[15] J. Cabot, R. Pau and R. Raventos, (2009), From UML/OCL to SBVR specifications: A challenging transformation, Information Systems 35 (2010), pp 417-440.
[16] Z. Struska, R. Pergl, (2009). BORM-points: Introduction and Results of Practical Testing. In: Enterprise Information Systems, Lecture Notes in Business Information Processing, vol.24, pp. 590-599.

Chapter 22

The OpenPonk Modeling Platform

[224] Uhnák, P.; Pergl, R. The OpenPonk Modeling Platform. In Proceedings of the 11th Edition of the International Workshop on Smalltalk Technologies, IWST'16, New York, NY, USA: ACM, 2016, ISBN 978-1-4503-4524-8, pp. 14:1–14:11.

The OpenPonk modeling platform

Peter Uhnák
Department of Software Engineering
Faculty of Information Technology
Czech Technical University in Prague
Czech Republic
uhnakpet@fit.cvut.cz

Robert Pergl
Department of Software Engineering
Faculty of Information Technology
Czech Technical University in Prague
Czech Republic
perglr@fit.cvut.cz

ABSTRACT

In this paper we present OpenPonk: a free, open-source, simple to use platform for developing tools for conceptual modeling: diagramming, DSLs, and algorithms operating on the models and diagrams, such as automatic layouting, model transformations, validations, etc.

This project differentiates itself from the current efforts by providing completely free and open-source live development environment, which is simple to learn, use, and extend.

There are already several plugins and extensions that bring several notations and algorithms, some of which are presented in this paper, alongside the overview of the core of the platform, and how they integrate with each other. We also present a comprehensive project case study utilizing OpenPonk.

CCS Concepts

•Software and its engineering → Integrated and visual development environments; •Human-centered computing → *Visualization systems and tools;* **•Computing methodologies →** Modeling methodologies;

Keywords

OpenPonk, modeling, visualizations, Pharo, DynaCASE, Roassal, UML, BORM

1. INTRODUCTION

In this paper we present OpenPonk (formerly known as DynaCASE) – an emerging modeling platform implemented in the live environment Pharo[2].

In all engineering endeavours, engineers utilize various types of diagrams that help them analyze and design their complex systems; civil engineers use CAD tools, software engineers use IDEs and CASE tools, and enterprise engineers use CABE tools. Also there are many research groups that focus on research in some of the aspects of these tools.

IWST'16 August 22-26th 2016, Prague, Czech Republic.

DOI: http://dx.doi.org/10.1145/2991041.2991055

Development of such tools is very demanding, because a lot of effort has to be invested into creating the foundation of the tool such as graphical visualization, interaction of graphical objects, persistence, layouting, and general user interface. To do these projects in high quality requires big budgets and teams. However, there are often small or mid-sized research groups and individual practitioners who have an idea that they would like to implement, whether their own modeling notation, specific algorithms, model transformations, simulations, etc. If they attempt to implement their tool from the scratch, the resulting product ends up often subpar and isolated from other tools, consequently not used in the end by a larger audience, thus wasting the ideas and invested resources. Additionally, resources that should have been invested into the research itself have to be wasted on reimplementing solved problems.

This is why we started the OpenPonk platform. We want to give designers of tools a platform which solves recurring tasks mentioned above, so they may focus on the core of their needs.

Our aim is not to replace or compete with existing industry-standard solutions for standardized notations, such as UML and Enterprise Architect, although there is some overlap of functionality. The main focus is to provide platform for tool building for the long tail of non-standard custom models and algorithms for both research and business.

The core of our tool and current extensions are available as open-source under the MIT license[1].

1.1 Goals and objectives of the paper

The purpose of this paper is to present the current state of OpenPonk. In the first part, we present a high-level overview of the *core* architecture and design behind OpenPonk. In the second part we overview the general approach and architecture of user-provided *plugins*, and how they connect to the core. In the third part, we present several plugins and extensions that we currently provide: simulating finite-state automata, modeling business entities using BORM, describing models with custom DSLs, and live model manipulation powered by MetaLinks. In the fourth part we present a use case study of using OpenPonk as a foundation for existing research efforts and the support for UML Class Diagrams with round-trip engineering for CORMAS. Finally, we discuss related solutions that are currently available.

2. ARCHITECTURE AND DESIGN

[1]https://openponk.github.io

In this section we present the core architecture of the system and high level view of the *Plugin Architecture*, which is explained in greater detail in the following section.

2.1 Terminology note

We further refer to the foundation of the platform as the *core*, and to user-supplied models, notations, and components as *plugins*.

Figures or *visual elements* are entities drawn on canvas, usually representing a particular model element.

Finally, we use the term *user* to refer to *user-developer*, i.e. a person that is using our platform as a basis to create or extend a plugin/model/notation. For a user that is simply using our platform and its extensions as a tool, we will use the term *tool-user*.

2.2 Connections between models, view, and controllers

OpenPonk is foremost a meta-modeling platform. Our aim is to provide the foundation for building modeling tools for creating and operating on a wide range of models, especially models that have a direct visual representation – *notation*. To represent the model and manipulate it through the notation, we in principle follow the model-view-controller (MVC) design [19], however due to possible model limitations we limit the direct interaction between the model and view and use controllers instead, as depicted in Figure 1.

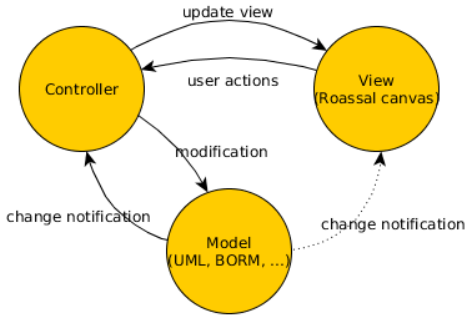


Figure 1: MVC design with limited communication between model and view

2.2.1 Model

The model is the most straightforward part, as it describes mere meta-models such as UML meta-model, BORM meta-model or Finite-State Automata meta-model.

2.2.2 View

The primary[2] view visualizes the notation of the meta-model. We have chosen Roassal visualization engine[1]. Roassal is a vector-based engine providing a support for a wide range of needs, such as chart and graph drawings, map visualizations, etc. It is also used by the Moose reengineering platform [13] for system complexity visualizations.

Importantly for us, Roassal provides a lower-level API for creating elementary shapes (e.g. ellipse, rectangle, line) and interactions (e.g. moving elements, resizing elements, zooming) that can be combined by the implementer to create appropriate notation elements. Furthermore, the core of Roassal is extensible. We created a range of new elements and interactions, many of which were contributed back to Roassal itself.

[2]Later we introduce secondary views, such as tree-views.

2.2.3 Controllers

Controllers are responsible for interpreting user-triggered signals coming from the view (adding a new figure, renaming an element, etc.) and for propagating updates to and from the model (meta-model instance). In the original MVC design[19], the view is directly observing the changes in the model and adopting them. We have diverged from this approach to handle non-observable models and to limit to the logic complexity in the view.

A lot of complex logic is hidden in the core controllers, thus simplifying the logic of the user-supplied ones.

2.2.4 Other components

The created models are organized in so-called *projects*. A project is a set of models and diagrams that are opened and stored together. The primary use case of projects is grouping together models describing various aspects of the modeled system.

OpenPonk additionally provides a set of prepared GUI components, and visual extensions that can enhance the produced tool.

3. PLUGIN ARCHITECTURE

The concrete meta-models and their notations are developed as so-called *plugins* by inheriting general classes of the core. Such plugins are independent of each other and can be distributed separately.

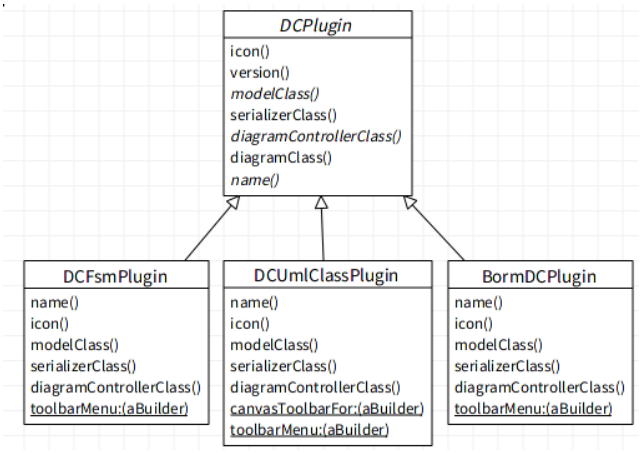


Figure 2: Plugin description classes

Basic properties of every plugin are described in a subclass of *Plugin* class as shown in Figure 2. At the moment, specifying only essential properties is necessary: the name of the plugin, the containment (top-level) meta-model class (i.e. the model that represents the whole diagram), and a controller class for the containment model. Optionally, the user can specify an icon (used in various places of the GUI), version of the plugin, and serializer used for file-system persistence.

3.1 Model creation

The meta-model infrastructure is the backbone of a plugin, and all other components[3] revolve around it. One of our requirements for the platform is the ability for the users to use their models. This means not only standard models such as UML, but also custom models developed by the user to address their specific business requirements.

A meta-model typically consists of a set of implemented classes that represent subjects from the modeled domain. Figure 3 shows an example model of a Finite-State Automata (FSM) meta-model.

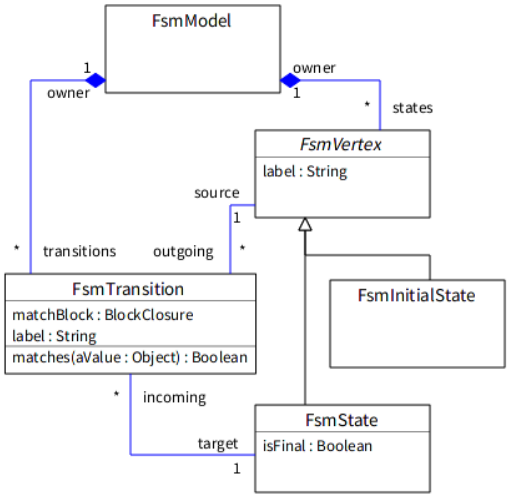


Figure 3: Example Finite-State Automata meta-model

Integrating models that have been developed independently of our tool however presents an integration challenge, as the model may inhibit some more advanced properties of the platform. More precisely, many components, e.g. the notation visualization in particular, require a mechanism through which changes in the model can be observed so that they can be updated accordingly. As we noted earlier, to address this need we delegate where possible the burden of updates to the controllers. Such an approach does inhibit some parts of the platform, as updates must go through the controller (using command pattern or through direct calls), or the user must explicitly inform the controller that changes to the model have been made. For the standard approach where the user creates and manipulates the model through the provided utilities (e.g. palette, canvas, form editor) this does not present a problem, as the interactions are automatically wrapped.

We use this approach for our UML plugin that utilizes the FAMIX[7] meta-model as the basis for the meta-model. Although we have extended the FAMIX infrastructure with additional classes required for UML, we have not modified the model itself; in fact neither was FAMIX created with our tool in mind, nor does it provide sufficient mechanisms for observing changes. Despite this, we were able to successfully create a diagramming plugin as the platform is powerful enough to handle the problems properly.

[3]e.g. visualization, simulation

Naturally, if the model does provide the necessary mechanism, not only can the implementation be simplified as some burden is removed from the controllers, but additional possibilities open up.

3.2 Creating visual elements

Meta-models that we are particularly interested in have an accompanying visual diagram notation.

Notation	Diagram elements
«keyword**»** **Classifier1** {abstract}	UMLClassifierShape (Classifier) - UMLKeywordLabel (Classifier) - UMLNameLabel (Classifier) - UMLLabel (Classifier) - UMLCompartment
attributes attribute1 inherited attribute2 ...	--- UMLLabel --- UMLLabel (Property) --- UMLLabel (Property, inherited) --- UMLLabel
operations operation1	- UMLCompartment --- UMLLabel --- UMLLabel (Operation)

Figure 4: Composition of visual elements

In Figure 4 we can see a UML Class composed from several *primitive* shapes as prescribed by UML Diagram Interchange[14].

The creation of the visual element is stored in the appropriate controller in the`#createFigure` method. The only requirement of the method is that the returned object understands `renderIn: aRoassalView`. That way, the method can both directly use Roassal API and return a Roassal element [Figure 5], or add an intermediate layer [Figure 6].

DynaCASE-DEMO-OCD-Controllers
OCDElementaryActorRoleController >> createFigureInView:

```
createFigureInView: aView
    | element |
    element := (RTBox new
        size: OCDDiagramController universalElementSize x;
        color: Color white;
        borderColor: Color black;
        borderWidth: 2) element.
    " ... "
    ^ element
```

Figure 5: Returning Roassal elements/shapes

When the model changes, the figure typically has to reflect the new properties of it. The view however is not completely redrawn when an update is required, instead only the concerned parts are updated. To tell the platform what and how should be updated, the user implements a *#refreshFigure* method in the accompanying controller. In this method updates such as changing text content, colors, adding and removing subelements, etc. can occur. The *#refreshFigure* is automatically called by the platform after a model change was detected, this can either be a result of the platform observing changes in the model (if possible), or when the model was modified through one of the platform's editing interfaces. With this approach the user does not need to concern themselves when an update should occur.

In addition, if the model provides sufficient observation granularity, the figure can directly observe the model and update accordingly.

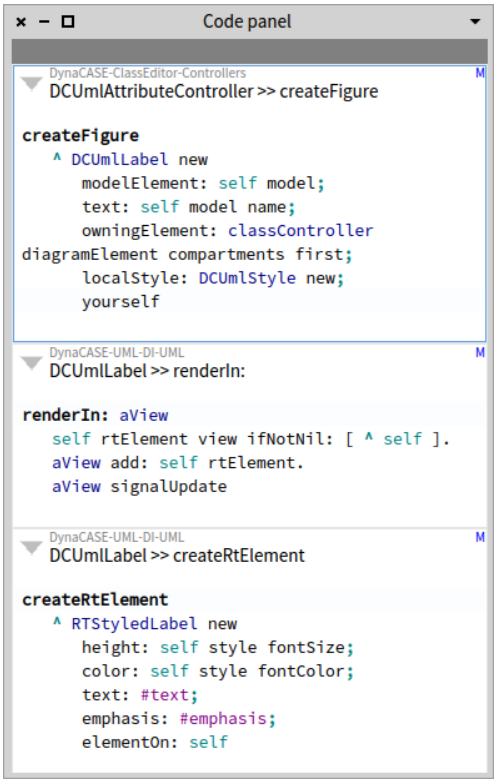


Figure 6: Using an intermediate layer

3.3 Controllers

Controllers are the glue between the model and views. The most common approach is to have a controller for each model element (e.g. a `Method` and a `MethodController`, `Classifier` and a `ClassifierController`, etc.). Depending on the appropriate granularity, additional responsibility may be taken (e.g. `MethodController` also handling `Parameter` model elements).

Apart from responsibility of creating the view described in the previous sections, two more UI-related interfaces are provided:

1. Each model element has typically a different set of properties that are modified by the tool-user. Therefore, each controller is free to override the `buildEditor:` method and to specify a custom Form consisting of the appropriate form elements (e.g. input text, droplist, checkbox), and the binding between the model and the form elements. This form automatically opens when the tool-users selects a model element.

2. Each diagram (notation) is usually accompanied by a different palette. The responsibility of the *Diagram Controller* (the master controller for the diagram) is to implement the palette specification [Figure 7]. The platform will then handle the actual creation of the appropriate objects when the tool-user selects one of the items and interacts with the canvas.

3.3.1 Connecting elements and live validation

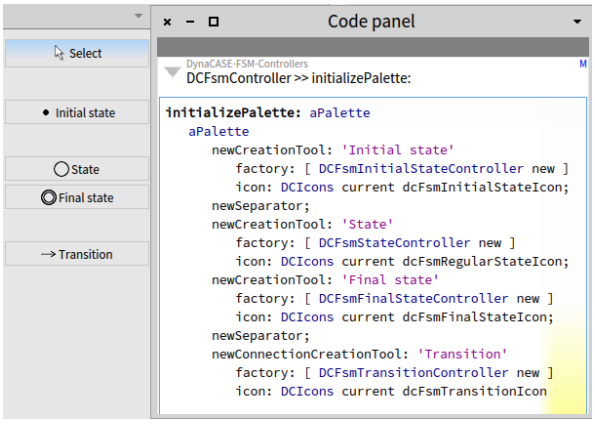


Figure 7: Palette specification

Models elements, and their visual representations rarely live by their own. Instead, they are connected through references or compositions. The connection is accommodated by four functions implemented in the controllers: *#canBeSourceFor:*, *#canBeTargetFor:*, *#addAsSourceFor:*, *#addAsTargetFor:*; the full signature being *receivingController* For: aNewController*.

The purpose of the *#canBe*For:* methods is to decide whether the receiving controller accepts the argument, if they return *false*, the connection cannot proceed and the *#addAs*For:* methods will not be called.

The *#addAs*For:* methods contain the behavior associated with connecting the elements. For containers only #TargetFor:* is required. If, however, the created element is a binary association (typically an edge), the first (source) controller will implement the #SourceFor:* methods, and the second (target) controller the #TargetFor:* methods.

In addition, the platform automatically uses the result of *#canBe*For:* to display visual feedback on top of the visual element, such as green or red overlay if the element can or cannot be connected [Figure 8].

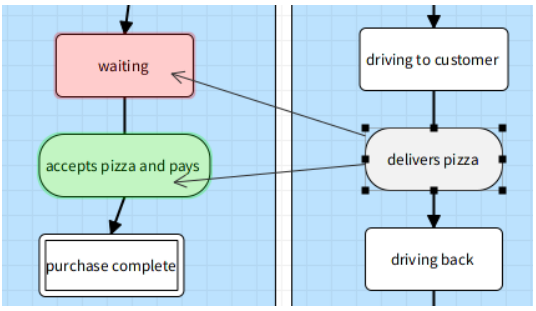


Figure 8: A communication can end in an activity (oval shape), but not in a state (rectangle shape) in BORM

Figure 9 shows that `StateController` can only be a target for a transition originating from an activity, whilst `ActivityController` can be also a target for a communication from an activity from a different participant (owner).

What may be unusual is that this validation does not fully rely on the model, instead the checks are made against the

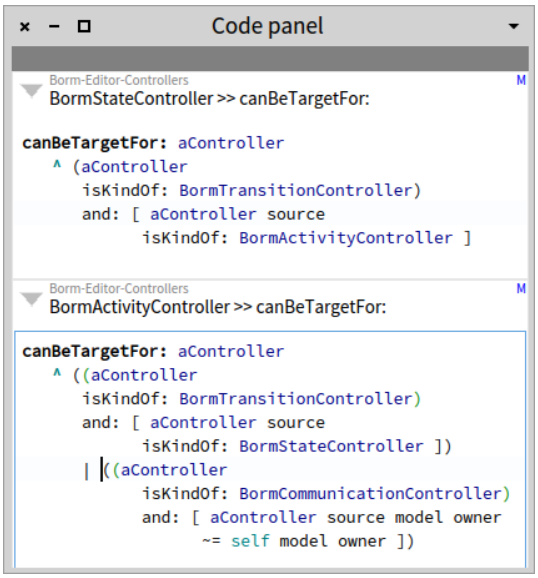


Figure 9: Code describing live validation

controllers. We have chosen this approach, as during more complex creation not all information may be available in the model, and the new model element is typically not yet connected with the present model, that is, sometimes we cannot decide whether an element can be connected only after it has been already connected. Thus complex meta-model structures can force the user to create only a partially valid model.

But the purpose of the live validation is not to always have fully valid model, instead it is a quick and cheap[4] to prevent common errors, similarly to a code editor warning a user of a missing semicolon or invalid syntax. For a full model validation, the user is free to implement a more powerful validation checker operating on the full model, as may be seen in Figure 10 for the case of an OntoUML validation editor[10].

4. EXTENSIONS AND NOTATIONS

We design the platform to be extensible and usable in different contexts and scenarios. To illustrate the wide possibilities, we present several quite different extensions that have been developed on top of the platform.

4.1 Model editing and live scripting

There are two principal ways in which a tool-user can modify meta-model instances. The first one is through the editors and tools provided by OpenPonk. This approach is common to majority of modeling tools (whether implemented in Java, C++, JavaScript, or Pharo). The popularity of this stems mainly from its ease-of-use, as the user is guided and hand-held by the tool's graphical interface, so the user does not accidentally corrupt the model. Moreover, the user does not need to be completely familiar with the actual meta-model implementation.

The second way is the ability to programatically manipulate the model, which offers interesting possibilities. Advanced modeling tools built in non-live environments (such as Java) address this by providing a special manipulation language that enables the user to query and manipulate the model, such as the Epsilon Object Language[5] for the Eclipse platform. There are, however, major downsides to this approach: implementing such a query language requires additional engineering effort on the part of the tool developer, while learning the query language requires additional effort on the part of the tool user. Furthermore, the language can be limiting in its capabilities and its usability, as without proper tooling support, aiding the user in creating, debugging, and working with the language may be limited.

We have mitigated this problem by choosing the Pharo live programming environment[2]. Pharo can be compared to a programming language, IDE, and an operating system rolled into one. In this platform, both the (tool-)user and the developer have direct access to objects in the system. This means that instead of using a specially crafted language, the user can directly use the Pharo language with all the powerful tools and support available in Pharo for regular development.

Naturally, in some contexts, providing an alternative way and a specific manipulation language may be beneficial, however by choosing Pharo we already have a powerful language applicable for all meta-models without any extra engineering effort required.

4.2 Observing model changes with MetaLinks

As we noted earlier, to be able to directly manipulate the meta-model instances and to be able to immediately see the changes in the diagram, the meta-model code base has to offer an API through which changes can be observed. Doing so, however, requires modification of the code, which in some cases may either not be possible, or it may introduce unreasonable overhead (e.g. from performance point of view).

To address this issue, one can make use of MetaLinks [4]. MetaLinks are a recent addition to Pharo that allows installing additional behavior to methods[5] without modifying the code itself. Therefore one can define code extensions that would announce changes to the model, and those extensions would be then installed to the methods on demand, without affecting the original code base. As the MetaLink's interface is very low level, we have developed an open-source toolkit that would simplify the most common use cases, by the name of *MetaLinks-Toolkit*[6]. Albeit this toolkit is still in its early stage, one can utilize it to ease the MetaLinks integration. Figure 11 shows a code installing observations to a code base. After the installation, when the specified attributes have been modified, the objects fire announcement instances that can be observed. We have experimentally used this toolkit on our BORM meta-model, however our aim is to provide such augmentations for the FAMIX meta-model.

4.3 Simulating Finite-State Automata

In this section, we demonstrate an unanticipated integration of a finite-state automata simulator (FSA).

When an integration is anticipated, an API can be implemented that properly handles the needs and abstractions for

[4]Requires no extra effort from the tool-user.

[5]To individual nodes of the method's abstract syntax tree.

[6]Available online: https://github.com/peteruhnak/metalinks-toolkit

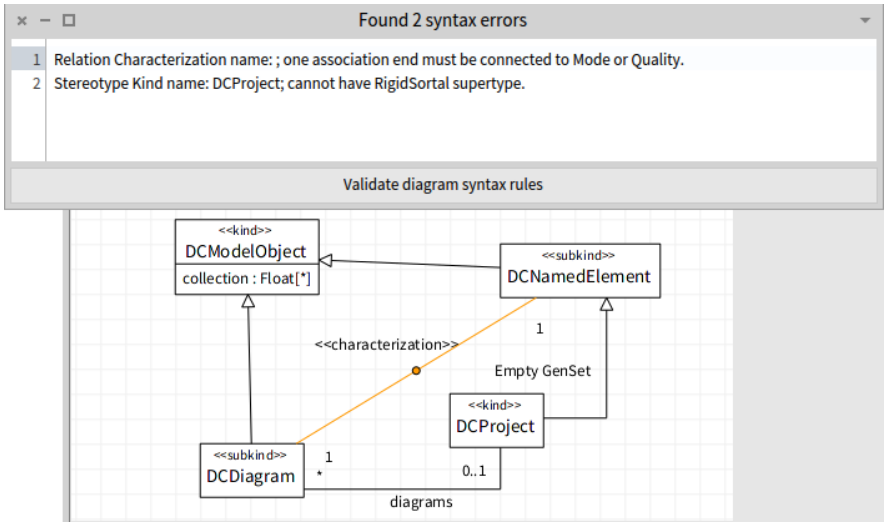


Figure 10: Full model validation of OntoUML diagrams

```
observations := {
    MTObservationSet
        target: MTElement
        change: #(name owner uuid)
        add: #()
        remove: #().

    MTObservationSet
        target: MTContainer
        change: #()
        add: #(add:)
        remove: #(remove:)
}.
ci := MTMetaLinksChangeInstaller new.
ci install: observations.
element := MTElement new.
container := MTContainer new.

"Now we can use normal Announcement propagation"
element when: ValueChanged do: [ :oldValue :newValue | ... ].
container when: ValueAdded do: [:newValue | ... ].
container when: ValueRemoved do: [:newValue | ... ].
```

Figure 11: Installing MetaLinks-based observations

the to-be-integrated tool. Handling all possible cases, however, requires many layers of abstraction, which not only have to be implemented on the platform-side, but, more importantly, every time a user wishes to create an extension. Thus, apart from solving the inherent complexity of the problem, the user is forced to address incidental complexity of the platform, as well.

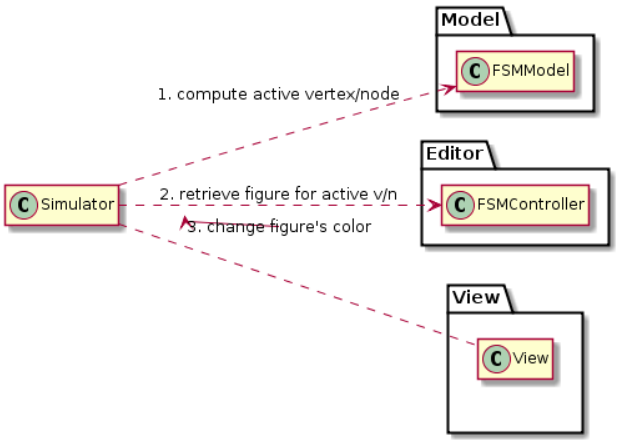


Figure 12: Simulator accessing different parts of the editor

The FSA simulator uses direct read-only access to the model and read/write access to the view. The model is used to compute the next step of the simulation, but no changes to the model are required. On the other hand, to visualize the progress through the simulation, the simulator directly modifies the view. It asks a diagram controller (responsible for managing the view) for an appropriate view element for the currently active node or edge and then changes the visualization (Figure 12) through the low-level Roassal API, or with the aid of existing OpenPonk-Roassal extensions.

In such an approach, both the FSA editor[7] and the OpenPonk platform are unaware of a third-party tool accessing their parts, therefore no explicit action is required of them (i.e. providing a special API).

Naturally, this approach requires the third-party tool to properly clean up after themselves, as the platform does not know what changes were made and what changes should be reverted.

Although such a third-party tool may inadvertently break the model or the view, we consider such risk acceptable, as this approach shortcuts otherwise complex implementation to bare essentials, which we find especially valuable for research and experimenting with and prototyping emerging ideas, which is one of the core aims OpenPonk. A more sophisticated and robust solution can always be introduced later.

4.4 DSLs for BORM and Class Diagrams

Although the tool-user creates a model primarily through the diagramming interface, there are other ways to create the model: importing from a different tool (via an exchange format such as XMI), transforming from other representation (reengineering source code), manually creating the required classes (via a programmable API), using a domain-specific language and possibly others. Choosing the best way is heavily context-dependent, therefore many tools will provide multiple options for the user.

To address this need, we have introduced a domain-specific language for two of our meta-models: BORM[9] and UML Class Diagrams[14].

Even though both models have visual notations, some users may prefer to describe their model using text, and possibly only adjust the layout of the resulting visualization. This is not a new approach and many tools provide this functionality, such as PlantUML[18]. Unfortunately, they often stop at the production of the visualization and there is no real model behind it, which the user can manipulate, transform, validate, etc. Our aim is to provide a DSL for model creation, not just visualization creation.

For the implementation of our DSLs we have chosen PetitParser[20]. PetitParser enables an easy creation of composable grammars by describing the grammar directly with Smalltalk code in a regex-like syntax.

For the need of editing the model via a DSL, we have introduced a DSL editor to the OpenPonk core. This editor is directly connected with the currently opened Workbench Editor, and the user can modify the model through it; saving the text in the editor updates the model, and refreshing the editor produces the textual representation of the model, as seen in Figure 13. The DSL editor synergizes with the classical diagramming interface, as the user can arbitrarily switch between them. As describing a model with text lacks certain diagram information, most notably the layout position, we have introduced a set of layouting algorithms to OpenPonk[16] that are automatically applied to the diagram; thus the user always sees a readable diagram instead of a stack of overlapping elements.

[7]The plugin(s) implementing the FSA meta-model and the notation.

5. PRACTICAL CASE STUDY: UML ROUND-TRIP ENGINEERING FOR ABM PLATFORM CORMAS

As our aim is to provide a platform for tool building, here we present a case study where OpenPonk was successfully used.

In collaboration with Cirad RU Green – a research group focused on addressing the needs of environmental research and sustainable agriculture – we have developed a UML Class Diagram Editor with round-trip engineering support for CORMAS agent-based modeling (ABM) platform[3] as a plugin for OpenPonk[17].

The users of CORMAS follow the participatory modeling (PM) approach in which the users (also known as stakeholders) collaborate on analyzing, creating, and implementing the domain model and the implementation solution.

For CORMAS, the stakeholders consist primarily of researchers and experts in the target domain; both, however, being seldom experienced programmers.

This presents a real challenge, as even though they are capable of creating the necessary models (the stakeholders pass a UML modeling course), they do not posses the sufficient programming knowledge and experience to correctly implement the model. For this reason, we have incorporated a support for round-trip engineering into our UML Class Diagram Editor. The objective is not to fully automate the process, but to aid with one of the challenging parts – creating the class structure from the model and vice versa.

The user can create the necessary model expressed in the UML notation with the diagram editor; for the needs of CORMAS we focus on a subset of UML notation typically used in class diagrams – classes, attributes, methods, associations, generalizations. The generator generates the necessary classes and methods in Pharo Smalltalk from the created model. Because the CORMAS platform is implemented in the VisualWorks Smalltalk (VW), we use the Smalltalk Interchange File Format[8] to exchange the source code between VisualWorks and Pharo, although direct code generation for VW has also been considered. For associations, we have chosen a straight-forward approach of providing accessors and add/remove methods[9], so that update on one side automatically updates the opposite sides. Other approaches have been researched [6]. These adhere more closely to the UML specifications, especially on maintaining visibility and multiplicity constraints of the associations; we found them needlessly complex in the context of the CORMAS platform, however for general-purpose forward engineering they may be suitable.

One of the important implemented features is the ability to reverse-engineer a model. In an ideal Model-Driven Development (MDD) scenario, only models are ever being manipulated and the code is always fully generated [11]. This, however, requires a very powerful modeling platform that is capable of describing every aspect of the software with models, which we currently do not provide. If there are modifications to the source code being made, the developers start to create discrepancies between the code and the model. It is possible to manually update the model each

[8]SIF. Available online: https://github.com/peteruhnak/sif/blob/master/c

[9]Add/remove methods are used for collection-based associations – association ends with multiplicities > 1

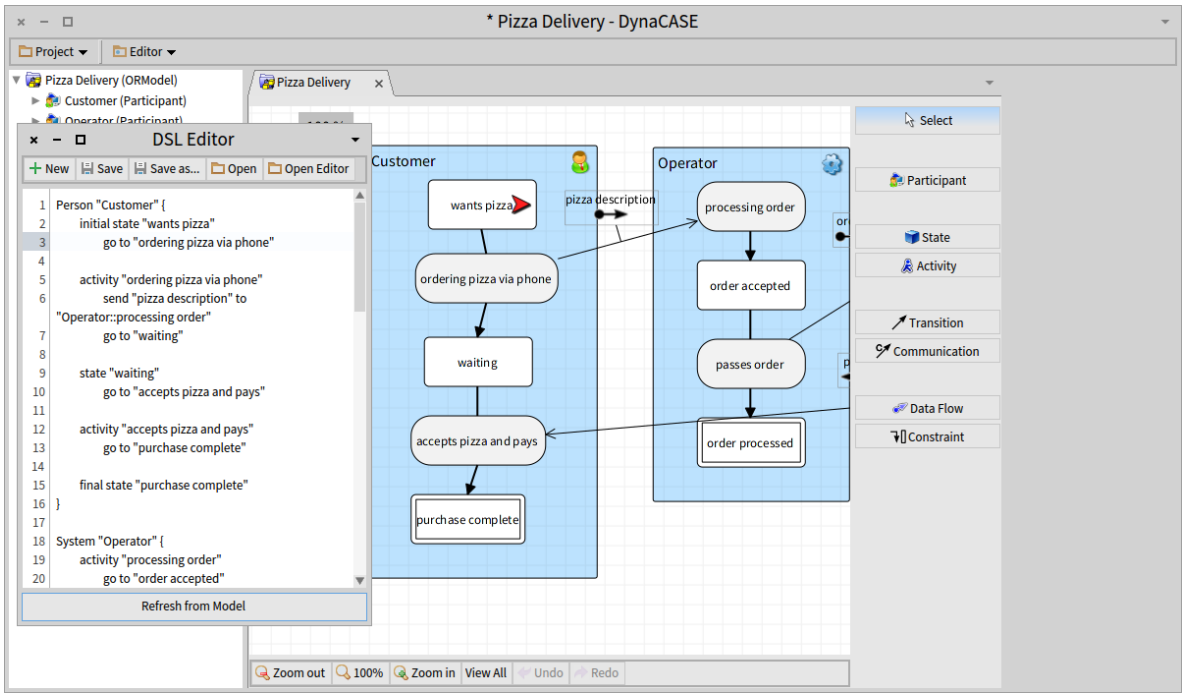


Figure 13: DSL editor on a BORM model

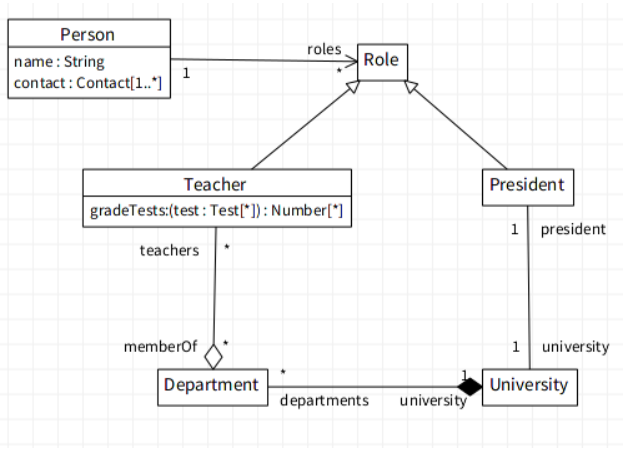


Figure 14: Example model

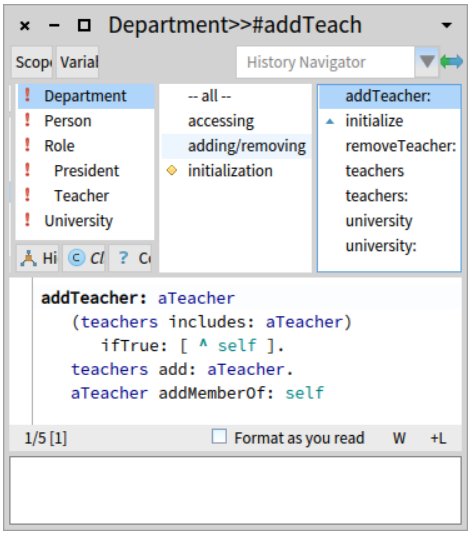


Figure 15: Classes and methods generated from the model

time the code has been made, that however poses the risk of complete abandonment of the original model if the discrepancies pile up beyond a threshold. Instead, we allow the developer to regenerate the model directly from the code.

Providing a generic reverse-engineering support is a challenging task, especially in dynamic languages, as the real types and constraints are known only during runtime, and reverse engineering certain constructs – associations in particular – may not be possible at all. As the models for CORMAS are not generic and therefore we can constrain the problem, we have provided an alternative solution. During the source code generation we add additional information to the source code — namely we add pragma annotations to the generated methods. They add meta-information that would be otherwise lost, as there is conceptual gap between the model and the code. With such approach during the reverse-engineering, we can reuse this meta-information to aid with the regenerated model. By putting the meta-model information directly alongside the related code, we also lower the probability of introducing discrepancies, as a developer modifying the code will be more likely to also update the meta-information because it is already part of the modified code.

Figure 16: A generated method with additional meta-information stored in a pragma

6. GRAPHICAL USER INTERFACE

The graphical user interface of the application is implemented using the Spec[22] framework.

Figure 17 shows the composition of GUI elements. The top-level window of the OpenPonk application is a *Workbench*. A workbench is always tied to a single project and has the responsibility of containing other parts of the GUI.

For each model in the project, an *Editor* can be opened within the workbench. The workbench organizes the editors in tabs, therefore several editors/tabs can be opened at once, although only one can be visible at a time.

The editor contains subwindows necessary for the display and manipulation of the model's notation – the *Canvas* (Roassal View) showing the visualization itself, a *Palette* providing a set of buttons for adding new items to the canvas, and an extensible bottom toolbar providing some manipulation buttons with easy access to the tool-user. Most of the GUI windows provide an API through which the user can manipulate them and describe their content.

Navigator: a tree-like view that visualizes the instances of meta-model elements in the project associated with the workbench; *Form*: a form editor for the currently selected meta-model instance capable of modifying its various properties; and finally a top toolbar that can be extended by plugins.

6.1 Connecting with plugins

Every window that is a part of the Workbench can be modified in some way by a plugin. Each window provides its own API appropriate to the situation.

The *Navigator* will ask the plugin's definition file about the structure of the model and how the model should be properly displayed, this consists primarily of specifying how to access the child nodes, element names, and icons as shown in Figure 18.

Both the *Workbench* menu (the toolbar in the top part of the Workbench), and *Editor* menu (underneath the canvas) can be extended. To extend any of the menus, a *pragma*[10] has to be added to a class-side method. OpenPonk will scan for methods containing the pragmas and call the methods when appropriate, as displaying the menu extensions can be restricted only show only if an editor of a particular plugin is selected. By choosing a pragma to implement this behavior, we have added extra flexibility as not only the plugin itself can customize the menus (the most common use case), but also any extra (or third-party) utility, without interfering with the plugin itself (e.g. a simulator window adding *"Open simulator"* button).

7. RELATED WORK

Several related works exist; we split them into three groups and address each individually: (i) stand-alone tools and environments, (ii) Eclipse-based tools, and (iii) Eclipse itself. The first group contains stand-alone tools and environments that do not rely on any outside solution. MetaEdit+ and Enterprise Architect are typical representants of this group.

Enterprise Architect (EA)[21] offers an industry-grade MDA solution for UML and UML-based models (such as BPMN). EA provides support for the complete development life-cycle including requirements engineering, modeling, source code generation, round-trip engineering, testing, and more. While OpenPonk has been created with similar use-cases in mind, we provide it for free, as open-source, and platform independent (EA supports only Windows). By choosing Pharo live environment, OpenPonk can also offer significantly more interactive options.

MetaCASE[12] provides a solution for designing custom domain-specific languages (concepts, rules, notations, code generators) with its MetaEdit+ Workbench aimed at expert developers, and a MetaEdit+ Modeler aimed at model users. The same comments hold as for EA.

The second group of tools includes a wide spectrum of modeling tools such as Modelio, Papyrus, OpenCABE (our original BORM tool[15]). These tools are created on the Eclipse platform through its Graphical Modeling Project (GMP), which includes Graphical Modeling Framework (GMF) and Graphical Editing Framwork (GEF) [8]. Such tools usually focus on a single model family and they are limited in their meta-modeling abilities.

Finally, the Eclipse platform itself (alongside its GMF and GEF frameworks) provides a strong foundation for modeling and meta-modeling tools – that is not only the development of the models themselves, but also the creation of additional

[10]Pragma is a method annotation that can be located in the system via reflectivity.

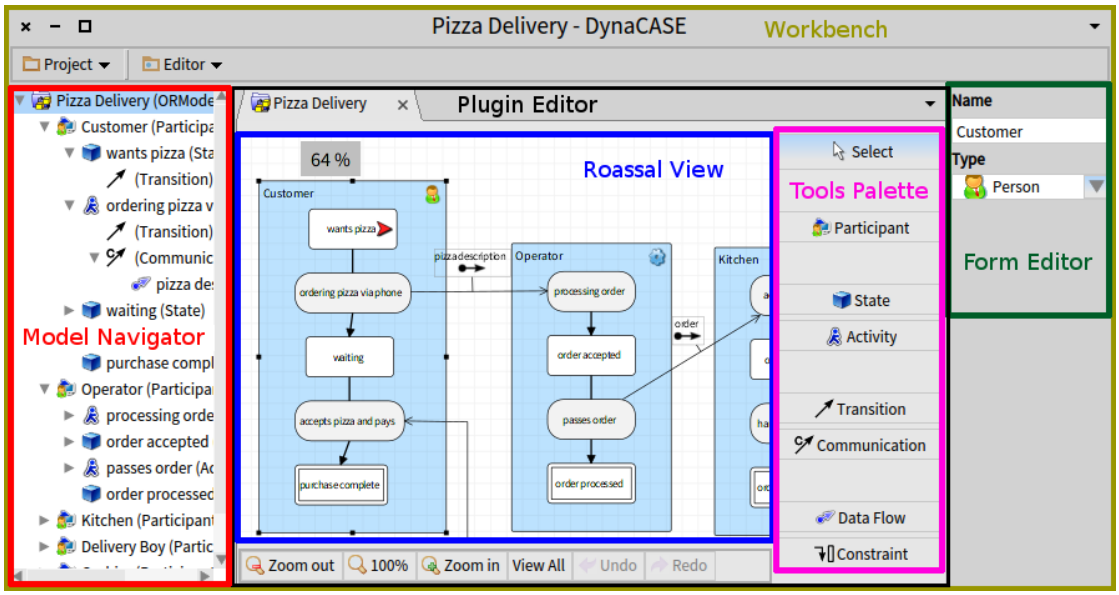


Figure 17: GUI overview

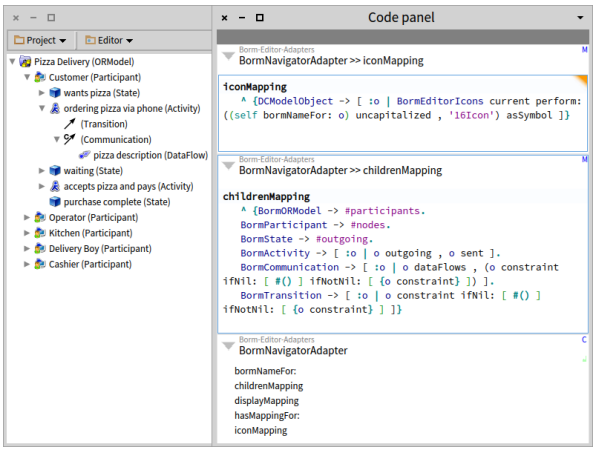


Figure 18: Specification of Navigator for BORM

tools and extensions operating on the models. Our OpenCABE[15] tool used for BORM modeling has been developed in Eclipse, so we have a very strong experience in this area. Eclipse platform is a very complex artefact with a steep learning curve. However, this would not be critical if the big investment in the development pays off in the form of resulting flexibility and ease of enhancements. Unfortunately, in our experience, it is not the case. On the contrary, the bigger the platform code base, the harder is to extend it with additional models, visualizations, simulations and other algorithms. After 6 years of development, we were not able to give the platform to students to easily implement their conceptual modeling ideas. With OpenPonk, we already had several successful student projects. As for the reason of this situation, we blame the *incidental complexity* of the Eclipse platform. As a more in-depth analysis is out of scope of this paper, let us just put a hypothesis that this incidental complexity comes from the limited dynamic and reflective possibilities of the Java platform; on the other hand, live, dynamic possibilities of Pharo enabled us to significantly limit the code base to the *inherent complexity* of the problem.

8. SUMMARY, CONCLUSION AND FUTURE WORK

OpenPonk project is the flagship of our research group, as we deal with various conceptual modeling notations, making models and performing algorithms on them. The project was initiated because of the lack of suitable free, open-source, simple, and extensible platform. The architecture of OpenPonk has been inspired by our extensive experience with the Eclipse platform. We took the best architectural ideas and stripped off the fat and gore by implementing it using the Pharo live programming environment.

The platform has been designed as highly modular – a minimal core extended by plugins and extensions. We have been also very keen about *separation of concerns* – model, view and controller are separated and various options for acquiring models are open: drawing diagrams, importing from an interchange file, reverse-engineering source code, DSL parsing, and other. The separation of concerns enables existing meta-models to be enhanced and used in OpenPonk without modifying their code.

OpenPonk stands for "dynamic": the meta-model and model development is highly interactive thanks to the live nature of Pharo. Querying and manipulating models on-the-fly is "for free".

OpenPonk is still in an early stage in many aspects. However, several quite diverse projects were successfully implemented using the platform, which demonstrate the possibilities. Several bachelor and master students were already able to acquire a working knowledge of OpenPonk and implement their ideas. This is very encouraging for us, as our ultimate goal is to offer a playground that will be loved by students, researchers and practitioners. With every new project, the

platform matures and offers more for everybody. The adoption by community is very important in this point and it will direct the future development. The development of the core itself focuses on providing a stable minimalistic, yet powerful, foundation for tools building via plugins development.

As for the current endeavours, we cooperate with ForMetis Enterprise Engineers, a Dutch consulting and development company, to implement simulations and validations for enterprise engineering models and we are in a close contact with INRIA Lille Nord Europe and the University of Antwerp, who are interested in cooperation.

9. ACKNOWLEDGMENTS

Currently, the development of OpenPonk is sponsored by ForMetis Consultants[11], our Dutch industrial partner. The development of the round-trip engineering support and partially the UML Class Editor for ABM CORMAS has been financed by RU Green CIRAD[12]. The development of the MetaLinks Toolkit has been sponsored by Synectique[13] and ESUG through their mobility support program[14].

10. REFERENCES

[1] Alexandre Bergel. *Agile Visualization*. 2016.

[2] A. Bergel, D. Cassou, S. Ducasse, J. Laval, and J. Bergel. *Deep into Pharo*. Square Bracket, [S.l], 2013.

[3] P. Bommel, N. Becu, C. Le Page, and F. Bousquet. Cormas, an Agent-Based simulation platform for coupling human decisions with computerized dynamics. 2015. https://agritrop.cirad.fr/576753/2/CormasforIsaga2015.pdf.

[4] M. Denker. *Sub-method Structural and Behavioral Reflection*. PhD thesis, University of Bern, 2008.

[5] Dimitris Kolovos, Louis Rose, Antonio Garcia-Dominguez, and Richard Paige. *The Epsilon Book*, volume 20. 2016.

[6] Dominik Gessenharter. Implementing UML associations in Java: a slim code pattern for a complex modeling concept. In *Proceedings of the Workshop on Relationships and Associations in Object-Oriented Languages*, RAOOL '09, pages 17–24, New York, NY, USA, 2009. ACM. 00008.

[7] S. Ducasse, N. Anquetil, M. U. Bhatti, A. C. Hora, J. Laval, and T. Girba. MSE and FAMIX 3.0: an interexchange format and source code model family. 2011.

[8] Eclipse. *Graphical Modeling Project*. 2016.

[9] Martin Podloucký and Robert Pergl. Towards Formal Foundations for BORM ORD Validation and Simulation. pages 315–322. SCITEPRESS - Science and and Technology Publications, 2014.

[10] Matúš Vološin. Vizualizace instancí OntoUML modelů. Diplomová práca. Praha: České vysoké účení technické v Praze, Fakulta informačních technologií, 2016.

[11] S. J. Mellor and M. J. Balcer. *Executable UML: a foundation for model-driven architecture*. Addison-Wesley, Boston ; San Francisco ; New York, 2002.

[12] MetaCase. MetaEdit+, 2016.

[13] O. Nierstrasz, S. Ducasse, and T. GÇŘrba. The story of Moose: an agile reengineering environment. *ACM SIGSOFT Software Engineering Notes*, 30(5):1–10, 2005.

[14] OMG. OMG Unified Modeling Language (UML) 2.5, Mar. 2015.

[15] R. Pergl and J. Tůma. OpenCASE âĂŞ a tool for ontology-centred conceptual modelling. In *Advanced Information Systems Engineering Workshops*, pages 511–518. Springer, 2012.

[16] Peter Uhnák. Layouting of Diagrams in the DynaCASE Tool. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

[17] Peter Uhnák and Pierre Bommel. Facilitating the Design of ABM and the Code Generation to Promote Participatory Modelling. 2016.

[18] PlantUML. PlantUML Language Reference Guide, 2016.

[19] S. T. Pope and G. E. Krasner. A Cookbook for Using Model-View-Controller User Interface Pradigm in Smalltalk-80. 1988.

[20] L. Renggli, S. Ducasse, T. GÃőrba, and O. Nierstrasz. Practical dynamic grammars for dynamic languages. In *4th Workshop on Dynamic Languages and Applications (DYLA 2010)*, 2010.

[21] Sparx Systems. Enterprise Architect, 2016.

[22] B. Van Ryseghem, S. Ducasse, and J. Fabry. Seamless composition and reuse of customizable user interfaces with Spec. *Science of Computer Programming*, 96:34–51, 2014.

[11] http://formetis.nl/
[12] http://ur-green.cirad.fr/
[13] http://synectique.eu/index.php
[14] http://esug.org/wiki/pier/Promotion/Mobility

Part III

References

Bibliography

[1] Alexander, C. *Notes on the Synthesis of Form*. Harvard University Press, 1964, ISBN 978-0-674-62751-2.

[2] Centre for Conceptual Modelling and Implementation. Centre for Conceptual Modelling and Implementation (CCMi): Research Group's Web Pages. Available from: `https://ccmi.fit.cvut.cz`

[3] Mylopoulos, J. Conceptual Modelling and Telos. In *Conceptual Modeling, Databases, and Case: An Integrated View of Information Systems Development*, 1992.

[4] Ullmann, S. *Semantics: an introduction to the science of meaning*. Barnes & Noble, Dec. 1978, ISBN 978-0-06-497076-1.

[5] Baldinger, K. *Semantic Theory: Towards a Modern Semantics*. New York: Palgrave Macmillan, Sept. 1980, ISBN 978-0-312-71258-7.

[6] Ogden, C. K.; Richards, I. A. *Meaning Of Meaning*. San Diego: Mariner Books, first edition, June 1989, ISBN 978-0-15-658446-3.

[7] Guizzardi, G. *Ontological Foundations for Structural Conceptual Models*, volume 015. Enschede: University of Twente, 2005, ISBN 90-75176-81-3.

[8] Stamper, R. *Information in business and administrative systems*. New York: Wiley, 1973, ISBN 978-0-471-82045-1.

[9] Mesarović, M. D.; Macko, D.; Takahara, Y. *Theory of Hierarchical, Multilevel, Systems*. Academic Press, 1970.

[10] Henderson-Sellers, B. *On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages*. Springer, Aug. 2012, ISBN 978-3-642-29825-7.

[11] Kelly, S.; Tolvanen, J.-P. Visual domain-specific modelling: Benefits and experiences of using metaCASE tools. In *International Workshop on Model Engineering, at ECOOP*, volume 2000, Citeseer, 2000.

[12] Gray, J.; Rossi, M.; Tolvanen, J.-P. Preface. *Journal of Visual Languages & Computing*, volume 15, no. 3, 2004: pp. 207 – 209, ISSN 1045-926X.

[13] Fowler, M. *Domain-Specific Languages*. Upper Saddle River, NJ: Addison-Wesley Professional, first edition, Oct. 2010, ISBN 978-0-321-71294-3.

[14] Verdonck, M.; Gailly, F.; de Cesare, S.; et al. Ontology-driven conceptual modeling: systematic literature mapping and review. *Applied Ontology*, volume 10, no. 3-4, Jan. 2015: pp. 197–227, ISSN 1570-5838, doi:10.3233/AO-150154.

[15] Smith, B.; Welty, C. Ontology—towards a New Synthesis. In *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, New York, NY, USA: ACM, 2001, ISBN 978-1-58113-377-6, pp. .3–.9, doi: 10.1145/505168.505201.

[16] Wolff, C. *Philosophia prima, sive ontologia*. Francofurti & Lipsiæ, prostat in Officina libraria Rengeriana, 1736. Available from: `http://archive.org/details/bub_gb_1HsPAAAAQAAJ`

[17] Stefik, M.; Conway, L. Towards the principled engineering of knowledge. *AI Magazine*, volume 3, no. 3, 1982: p. 4.

[18] Brachman, R. J.; Schmolze, J. G. An overview of the KL-ONE knowledge representation system. In *Readings in Artificial Intelligence and Databases*, Elsevier, 1988, pp. 207–230.

[19] Studer, R.; Benjamins, V. R.; Fensel, D.; et al. Knowledge engineering: principles and methods. *Data and knowledge engineering*, volume 25, no. 1, 1998: pp. 161–198.

[20] Jardine, D. A. *The ANSI/SPARC DBMS Model; Proceedings of the Second Share Working Conference on Data Base Management Systems, Montreal, Canada, April 26-30, 1976*. Elsevier Science Inc., 1977.

[21] Chen, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, volume 1, no. 1, 1976: pp. 9–36.

[22] Gašević, D.; Kaviani, N.; Milanović, M. Ontologies and software engineering. In *Handbook on Ontologies*, Springer, 2009, pp. 593–615.

[23] Calero, C.; Ruiz, F.; Piattini, M. *Ontologies for software engineering and software technology*. Springer Science & Business Media, 2006.

[24] Siricharoen, W. V. Ontologies and Software Engineering. In *International Conference on Computational Science*, Springer, 2007, pp. 1155–1161.

[25] Uschold, M.; King, M.; Moralee, S.; et al. The enterprise ontology. *The knowledge engineering review*, volume 13, no. 1, 1998: pp. 31–89.

[26] Dietz, J. L. G. Enterprise Ontology - Understanding the Essence of Organizational Operation. In *Enterprise Information Systems VII*, Dordrecht: Springer Netherlands, 2006, ISBN 978-1-4020-5323-8, pp. 19–30.

[27] Guarino, N.; Oberle, D.; Staab, S. What is an ontology? In *Handbook on ontologies*, Springer, 2009, pp. 1–17.

[28] Uschold, M.; Gruninger, M. Ontologies and semantics for seamless connectivity. *ACM SIGMod Record*, volume 33, no. 4, 2004: pp. 58–64.

[29] Collier, A. *Critical Realism: An Introduction to Roy Bhaskar's Philosophy*. London; New York: Verso, Apr. 1994, ISBN 978-0-86091-602-4.

[30] Guizzardi, G.; Halpin, T. Ontological Foundations for Conceptual Modelling. *Appl. Ontol.*, volume 3, no. 1-2, Jan. 2008: pp. 1–12, ISSN 1570-5838.

[31] Mealy, G. H. Another Look at Data. In *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, AFIPS '67 (Fall), New York, NY, USA: ACM, 1967, pp. 525–534, doi:10.1145/1465611.1465682.

[32] Kent, W.; Hoberman, S. *Data and Reality: A Timeless Perspective on Perceiving and Managing Information in Our Imprecise World, 3rd Edition*. Westfield, N.J.: Technics Publications, LLC, third edition edition, Feb. 2012, ISBN 978-1-935504-21-4.

[33] Weber, R. *Ontological foundations of information systems*. no. 4, Melbourne, Vic: Coopers & Lybrand and the Accounting Association of Australia and New Zealand, 1997.

[34] Heller, B.; Herre, H. Ontological Categories in GOL. *Axiomathes*, volume 14, no. 1, Mar. 2004: pp. 57–76, ISSN 1122-1151, 1572-8390, doi:10.1023/B:AXIO.0000006788.44025.49.

[35] Gangemi, A.; Guarino, N.; Masolo, C.; et al. Sweetening Ontologies with DOLCE. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Oct. 2002, ISBN 978-3-540-44268-4 978-3-540-45810-4, pp. 166–181, doi:10.1007/3-540-45810-7_18.

[36] Guizzardi, G.; Herre, H.; Wagner, G. On the general ontological foundations of conceptual modeling. In *International Conference on Conceptual Modeling*, Springer, 2002, pp. 65–78.

[37] Guizzardi, G.; Herre, H.; Wagner, G. Towards Ontological Foundations for UML Conceptual Models. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, number 2519 in Lecture Notes in Computer Science, Springer

Berlin Heidelberg, Oct. 2002, ISBN 978-3-540-00106-5 978-3-540-36124-4, pp. 1100–1117.

[38] Guizzardi, G.; Wagner, G.; Guarino, N.; et al. An ontologically well-founded profile for UML conceptual models. In *International Conference on Advanced Information Systems Engineering*, Springer, 2004, pp. 112–126.

[39] Guizzardi, G.; Wagner, G. Conceptual simulation modeling with Onto-UML. In *Proceedings of the Winter Simulation Conference*, Winter Simulation Conference, 2012, p. 5.

[40] Guizzardi, G. Logical, ontological and cognitive aspects of object types and cross-world identity with applications to the theory of conceptual spaces. In *Applications of Conceptual Spaces*, Springer, 2015, pp. 165–186.

[41] Guizzardi, G. Modal Aspects of Object Types and Part-Whole Relations and the de re/de dicto Distinction. In *International Conference on Advanced Information Systems Engineering*, Springer, 2007, pp. 5–20.

[42] Guizzardi, G. The Problem of Transitivity of Part-Whole Relations in Conceptual Modeling Revisited. In *Proccedings of 21st International Conference on Advanced Information Systems Engineering (CAISE'09)*, Amsterdam, The Netherlands, 2009.

[43] Guizzardi, G. On the Representation of Quantities and Their Parts in Conceptual Modeling. In *Proceedings of the 2010 Conference on Formal Ontology in Information Systems*, Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010, ISBN 978-1-60750-534-1, pp. 103–116.

[44] Guizzardi, G. Ontological Foundations for Conceptual Part-Whole Relations: The Case of Collectives and Their Parts. In *Advanced Information Systems Engineering*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, June 2011, ISBN 978-3-642-21639-8 978-3-642-21640-4, pp. 138–153, doi:10.1007/978-3-642-21640-4_12.

[45] Guizzardi, G.; Wagner, G.; Herre, H. On the foundations of uml as an ontology representation language. In *International Conference on Knowledge Engineering and Knowledge Management*, Springer, 2004, pp. 47–62.

[46] Guizzardi, G.; Masolo, C.; Borgo, S. In the Defense of a Trope-Based Ontology for Conceptual Modeling: An Example with the Foundations of Attributes, Weak Entities and Datatypes, 25th Intl. In *Conf. on Conceptual Modeling, Berlin*, 2006.

[47] Guizzardi, G.; Zamborlini, V. Using a trope-based foundational ontology for bridging different areas of concern in ontology-driven conceptual modeling. *Science of Computer Programming*, volume 96, 2014: pp. 417–443.

[48] Albuquerque, A.; Guizzardi, G. An ontological foundation for conceptual modeling datatypes based on semantic reference spaces. In *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, IEEE, 2013, pp. 1–12.

[49] Guizzardi, G.; Wagner, G. What's in a relationship: an ontological analysis. In *International Conference on Conceptual Modeling*, Springer, 2008, pp. 83–97.

[50] Costal, D.; Gómez, C.; Guizzardi, G. Formal semantics and ontological analysis for understanding subsetting, specialization and redefinition of associations in UML. In *International Conference on Conceptual Modeling*, Springer, 2011, pp. 189–203.

[51] Guarino, N.; Guizzardi, G. "We Need to Discuss the Relationship": Revisiting Relationships as Modeling Constructs. In *Advanced Information Systems Engineering*, Springer, 2015, pp. 279–294.

[52] Guizzardi, G. Agent Roles, Qua Individuals and the Counting Problem. In *Software Engineering for Multi-Agent Systems IV*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, May 2005, ISBN 978-3-540-33580-1 978-3-540-33583-2, pp. 143–160, doi:10.1007/11738817_9.

[53] Masolo, C.; Guizzardi, G.; Vieu, L.; et al. Relational roles and qua-individuals. *AAAI Fall Symposium - Technical Report*, Jan. 2005.

[54] Guizzardi, G.; Wagner, G.; Falbo, R. d. A.; et al. Towards Ontological Foundations for the Conceptual Modeling of Events. In *Conceptual Modeling*, number 8217 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, Jan. 2013, ISBN 978-3-642-41923-2 978-3-642-41924-9, pp. 327–341.

[55] Guizzardi, G.; de Almeida Falbo, R.; Guizzardi, R. S. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. In *CIbSE*, 2008, pp. 127–140.

[56] Guizzardi, R. S.; Guizzardi, G. Ontology-based transformation framework from TROPOS to AORML. *Social modeling for requirements engineering*, 2010: pp. 547–570.

[57] Falbo, R. d. A.; Quirino, G. K.; Nardi, J. C.; et al. An Ontology Pattern Language for Service Modeling. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, volume 2016, Pisa, Italy: ACM, Jan. 2016, ISBN 978-1-4503-3739-7, pp. 321–326, doi:10.1145/2851613.2851840.

[58] Griffo, C.; Almeida, J. P. A.; Guizzardi, G. Towards a legal core ontology based on Alexy's theory of fundamental rights. In *Multilingual Workshop on Artificial Intelligence and Law, ICAIL*, 2015.

[59] Barcellos, M. P.; de Almeida Falbo, R.; Frauches, V. Towards a Measurement Ontology Pattern Language. In *ONTO. COM/ODISE@ FOIS*, 2014.

[60] Guizzardi, G.; Wagner, G.; Almeida, J. P. A.; et al. Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story. *Applied Ontology*, volume 10, no. 3-4, Jan. 2015: pp. 259–271, ISSN 1570-5838, doi:10.3233/AO-150157.

[61] Vejrazkova, Z.; Meshkat, A. Translating DEMO Models into Petri Net. In *Enterprise and Organizational Modeling and Simulation*, volume 153, Springer Verlag Heidelberg, 2013, ISBN 978-3-642-41637-8.

[62] OMG. Unified Modeling Language, version 2.5.1. Dec. 2017. Available from: `https://www.omg.org/spec/UML/2.5.1`

[63] Arlow, J.; Neustadt, I. *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition)*. Addison-Wesley Professional, 2005, ISBN 0-321-32127-8.

[64] Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, third edition, Sept. 2003, ISBN 0-321-19368-7.

[65] Fuentes-Fernández, L.; Vallecillo-Moreno, A. An introduction to UML profiles. *UML and Model Engineering*, volume 2, 2004: pp. 6–13.

[66] OMG. System Modeling Language, version 1.5. May 2017. Available from: `https://www.omg.org/spec/SysML/1.5/`

[67] Benevides, A. B. *A Model-based Graphical Editor for Supporting the Creation, Verification and Validation of OntoUML Conceptual Models*. Master's thesis, UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, May 2010.

[68] Barbier, F.; Henderson-Sellers, B.; Le Parc-Lacayrelle, A.; et al. Formalization of the Whole-Part relationship in the Unified Modeling Language. *IEEE Transactions on Software Engineering*, volume 29, no. 5, May 2003: pp. 459–470, ISSN 0098-5589, doi:10.1109/TSE.2003.1199074.

[69] Krogstie, J. UML and the Unified Process. Hershey, PA, USA: IGI Global, 2003, ISBN 978-1-931777-44-5, pp. 1–22.

[70] Guizzardi, G. Representing Collectives and Their Members in UML Conceptual Models: An Ontological Analysis. In *Advances in Conceptual Modeling – Applications and Challenges*, number 6413 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, Jan. 2010, ISBN 978-3-642-16384-5 978-3-642-16385-2, pp. 265–274.

[71] OMG. Object Constraint Language, version 2.4. Feb. 2014. Available from: `https://www.omg.org/spec/OCL/2.4/`

[72] Richters, M.; Gogolla, M. OCL: Syntax, Semantics, and Tools. In *Object Modeling with the OCL, The Rationale Behind the Object Constraint Language*, London, UK, UK: Springer-Verlag, 2002, ISBN 978-3-540-43169-5, pp. 42–68.

[73] Cabot, J.; Teniente, E. Transformation techniques for OCL constraints. *Science of Computer Programming*, volume 68, no. 3, Oct. 2007: pp. 179–195, ISSN 0167-6423, doi:10.1016/j.scico.2007.05.001.

[74] Documents Associated with Business Process Model and Notation (BPMN) Version 2.0. Jan. 2015. Available from: `http://www.omg.org/spec/BPMN/2.0/`

[75] OMG. BPMN 2.0 by Example. 2010. Available from: `https://www.omg.org/cgi-bin/doc?dtc/10-06-02`

[76] Silver, B. *BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide: A structured approach for business process modeling and implementation using BPMN 2.0*. Cody-Cassidy Press, Oct. 2011, ISBN 0-9823681-1-9.

[77] Guizzardi, G.; Wagner, G. Can BPMN be used for making simulation models? *Lecture Notes in Business Information Processing*, volume 88 LNBIP, 2011: pp. 100–115.

[78] Dietz, J. L. G.; Hoogervorst, J. A. P.; Albani, A.; et al. The discipline of enterprise engineering. *International Journal of Organisational Design and Engineering*, volume 3, no. 1, 2013: pp. 86–114.

[79] Hevner, A. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, volume 19, no. 2, Jan. 2007.

[80] Dietz, J.; Hoogervorst, J. Enterprise Engineering Theories. Available from: `https://www.researchgate.net/project/Enterprise-Engineering-Theories`

[81] March, S. T.; Smith, G. F. Design and natural science research on information technology. *Decision Support Systems*, volume 15, no. 4, Dec. 1995: pp. 251–266, ISSN 0167-9236, doi:10.1016/0167-9236(94)00041-2.

[82] Hevner, A. R.; March, S. T.; Park, J.; et al. Design Science in Information Systems Research. *MIS Q.*, volume 28, no. 1, Mar. 2004: pp. 75–105, ISSN 0276-7783.

[83] Simon, H. A. *The Sciences of the Artificial - 3rd Edition*. Cambridge, Mass: The MIT Press, third edition, Oct. 1996, ISBN 978-0-262-69191-8.

[84] Dietz, J. L. G.; Hoogervorst, J. A. P. A critical investigation of TOGAF - based on the enterprise engineering theory and practice. In *Advances in Enterprise Engineering V*, Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg, May 2011, ISBN 978-3-642-21057-0 978-3-642-21058-7, pp. 76–90, doi: 10.1007/978-3-642-21058-7_6.

[85] Ettema, R.; Dietz, J. L. G. ArchiMate and DEMO – Mates to Date? In *Advances in Enterprise Engineering III*, volume 34, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN 978-3-642-01914-2 978-3-642-01915-9, pp. 172–186.

[86] Wieringa, R. J. *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ISBN 978-3-662-43838-1 978-3-662-43839-8.

[87] Op 't Land, M.; Dietz, J. L. G. Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations. In *Advances in Enterprise Engineering VI*, volume 110, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-29902-5 978-3-642-29903-2, pp. 77–92.

[88] Dietz, J. L. G. *Architecture: building strategy into design*. The Hague, The Netherlands: Academic Service, 2008, ISBN 978-90-12-58086-1.

[89] Terlouw, L. I.; Albani, A. An Enterprise Ontology-Based Approach to Service Specification. *IEEE Transactions on Services Computing*, volume 6, no. 1, 2013: pp. 89–101, ISSN 1939-1374, doi:10.1109/TSC.2011.38.

[90] Kervel, S. J. V.; Hintzen, J.; van Meeuwen, T.; et al. A professional case management system in production, modeled and implemented using DEMO. In *Complementary proceedings of the 8th Workshop on Transformation & Engineering of Enterprises (TEE 2014), and the 1st International Workshop on Capability-oriented Business Informatics (CoBI 2014)*, volume 1182, Geneva, Switzerland: Technical University of Aachen, July 2014, ISBN 1613-0073.

[91] Guerreiro, S.; van Kervel, S. J. H.; Vasconcelos, A.; et al. Executing Enterprise Dynamic Systems Control with the Demo Processor: The Business Transactions Transition Space Validation. In *Knowledge and Technologies in Innovative Information Systems*, volume 129, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33243-2 978-3-642-33244-9, pp. 97–112.

[92] Van Kervel, S.; Dietz, J.; Hintzen, J.; et al. Enterprise Ontology driven software engineering. In *ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends*, 2012, pp. 205–210.

[93] CIAO! Enterprise Engineering Network. CIAO! Enterprise Engineering Network web portal. Available from: `http://ciaonetwork.org/`

[94] Aalst, W. v. d. *Process Mining: Data Science in Action*. Berlin Heidelberg: Springer-Verlag, second edition, 2016, ISBN 978-3-662-49850-7.

[95] Knott, R.; Merunka, V.; Polak, J. The BORM methodology: a third-generation fully object-oriented methodology. *Knowledge-Based Systems*, volume 16, no. 2, Mar. 2003: pp. 77–89, ISSN 09507051, doi:10.1016/S0950-7051(02)00075-8.

[96] Struska, Z.; Merunka, V. BORM points - New concept proposal of complexity estimation method. In *ICEIS 2007 Proceedings*, Portugal: INSTICC-INST, 2007, ISBN 978-972-8865-90-0, pp. 580–586.

[97] Brožek, J.; Merunka, V.; Merunková, I. *Organization modeling and simulation using BORM approach*, *Lecture Notes in Business Information Processing*, volume 63 LNBIP. 2010.

[98] Merunka, V.; Merunková, I. Role of OBA Approach in Object-Oriented Process Modelling and Simulation. In *Enterprise and Organizational Modeling and Simulation*, number 153 in Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2013, ISBN 978-3-642-41637-8 978-3-642-41638-5, pp. 74–84.

[99] Knott, R.; Merunka, V.; Polak, J. Process modeling for object oriented analysis using BORM Object Behavioral Analysis. In *4th International Conference on Requirements engineering, 2000. Proceedings*, 2000, pp. 7 –16, doi:10.1109/ICRE.2000.855566.

[100] Merunka, V.; Nouza, O.; Brožek, J. Automated Model Transformations Using the C.C Language. In *Advances in Enterprise Engineering I*, *Lecture Notes in Business Information Processing*, volume 10, Springer Berlin Heidelberg, 2008, ISBN 978-3-540-68643-9, pp. 137–151.

[101] Frank, U. Multi-perspective enterprise modeling (MEMO) conceptual framework and modeling languages. IEEE Comput. Soc, 2002, ISBN 978-0-7695-1435-2, pp. 1258–1267, doi:10.1109/HICSS.2002.993989.

[102] Frank, U. Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design. *Business & Information Systems Engineering*, volume 6, no. 6, 2014: pp. 319–337, ISSN 2363-7005, 1867-0202, doi: 10.1007/s12599-014-0350-4.

[103] Henderson-Sellers, B.; Gonzalez-Perez, C. The rationale of powertype-based metamodelling to underpin software development methodologies. In *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling-Volume 43*, Australian Computer Society, Inc., 2005, pp. 7–16.

[104] Gonzalez-Perez, C.; Henderson-Sellers, B. A powertype-based metamodelling framework. *Software & Systems Modeling*, volume 5, no. 1, 2006: pp. 72–90.

[105] Guizzardi, G.; Almeida, J. P. A.; Guarino, N.; et al. Towards an Ontological Analysis of Powertypes. Jan. 2015.

[106] Carvalho, V. A.; Almeida, J. P. A.; Fonseca, C. M.; et al. Extending the Foundations of Ontology-Based Conceptual Modeling with a Multi-level Theory. In *Conceptual Modeling*, Lecture Notes in Computer Science, Springer, Cham, Oct. 2015, ISBN 978-3-319-25263-6 978-3-319-25264-3, pp. 119–133, doi:10.1007/978-3-319-25264-3_9.

[107] Carvalho, V. A.; Almeida, J. P. A.; Guizzardi, G. Using a Well-Founded Multi-level Theory to Support the Analysis and Representation of the Powertype Pattern in Conceptual Modeling. In *Advanced Information Systems Engineering*, volume 9694, Cham: Springer International Publishing, 2016, ISBN 978-3-319-39695-8 978-3-319-39696-5, pp. 309–324, doi:10.1007/978-3-319-39696-5_19.

[108] Carvalho, V. A.; Almeida, J. P. A.; Fonseca, C. M.; et al. Multi-level Ontology-based Conceptual Modeling. *Data Knowl. Eng.*, volume 109, no. C, May 2017: pp. 3–24, ISSN 0169-023X, doi:10.1016/j.datak.2017.03.002.

[109] Carvalho, V. A.; Almeida, J. P. Toward a Well-founded Theory for Multi-level Conceptual Modeling. *Softw. Syst. Model.*, volume 17, no. 1, Feb. 2018: pp. 205–231, ISSN 1619-1366, doi:10.1007/s10270-016-0538-9.

[110] Clark, T.; Sammut, P.; Willans, J. S. Applied Metamodelling: A Foundation for Language Driven Development (Third Edition). *CoRR*, volume abs/1505.00149, 2015.

[111] Clark, T.; Sammut, P.; Willans, J. S. Super-Languages: Developing Languages and Applications with XMF (Second Edition). *CoRR*, volume abs/1506.03363, 2015.

[112] Bock, A.; Frank, U. Multi-perspective Enterprise Modeling—Conceptual Foundation and Implementation with ADOxx. In *Domain-Specific Conceptual Modeling*, Springer, Cham, 2016, ISBN 978-3-319-39416-9 978-3-319-39417-6, pp. 241–267.

[113] Schmidt, D. C. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, volume 39, no. 2, 2006: p. 25.

[114] Sitiol, A. CASE technology. In *Student Conference on Research and Development, 2002. SCOReD 2002*, 2002, pp. 54–57, doi:10.1109/SCORED.2002.1033053.

[115] Kleppe, A.; Warmer, J.; Bast, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston: Addison-Wesley Professional, first edition, May 2003, ISBN 978-0-321-19442-8.

[116] OMG. MDA Guide Revision 2.0. 2014. Available from: `http://www.omg.org/cgi-bin/doc?ormsc`

[117] Rybola, Z. *Towards OntoUML for Software Engineering: Transformation of OntoUML into Relational Databases*. Ph.D., Czech Technical University in Prague, Prague, Czech Republic, Aug. 2017.

[118] Richta, K.; Rybola, Z. Transformation of Relationships from UML/OCL to SQL. In *ITAT 2011: Zborník príspevkov prezentovaných na konferencii ITAT*, volume 11, Terchová, Slovakia: University of P. J. Šafárik, Košice, Slovakia, Sept. 2011, ISBN 978-80-89557-01-1.

[119] Rybola, Z. Constraint for Multiplicities of Binary Relationships. In *POSTER 2011*, volume 15, Prague, Czech Republic: Faculty of Electrical Engineering, Czech Technical University in Prague, May 2011, ISBN 978-80-01-04806-1.

[120] Rybola, Z.; Richta, K. Transformation of Special Multiplicity Constraints - Comparison of Possible Realizations. In *FedCSIS 2012*, Wroclaw, Poland, Sept. 2012.

[121] Rybola, Z.; Richta, K. Using OCL in Model Validation According to Stereotypes. In *DATESO 2012*, volume 12, Žernov, Rovensko pod Troskami, Czech Republic, Apr. 2012, ISBN 978-80-7378-171-2, pp. 93–102.

[122] Rybola, Z.; Richta, K. Validation of stereotypes' usage in UML class model by generated OCL constraints. In *Information Technologies - Applications and Theory 2012*, Bielanské Tatry, Slovakia: Technical University of Košice, Sept. 2012, ISBN 978-80-971144-1-1, pp. 25–32.

[123] Kervel, S. J. H. v. *Ontology driven Enterprise Information Systems Engineering*. dissertation thesis, Technische Universiteit Delft, Netherlands, 2012.

[124] Guerreiro, S.; Van Kervel, S.; Babkin, E. Towards devising an architectural framework for enterprise operating systems. In *ICSOFT 2013 - Proceedings of the 8th International Joint Conference on Software Technologies*, 2013, pp. 578–585.

[125] Victor, B. Explorable explanations. 2011. Available from: `http://worrydream.com/ExplorableExplanations`

[126] Victor, B. Inventing on principle. 2012. Available from: `https://www.youtube.com/watch?v=PUv66718DII`

[127] Victor, B. Learnable programming: Designing a programming system for understanding programs. 2012. Available from: `http://worrydream.com/LearnableProgramming`

[128] Victor, B. Media for Thinking the Unthinkable. 2013. Available from: `http://worrydream.com/MediaForThinkingTheUnthinkable/`

[129] Jason Gilman. Proto REPL, a New Clojure Development and Visualization Tool. 2016. Available from: `https://www.youtube.com/watch?v=buPPGxOnBnk`

[130] Victor, B. The Future of Programming. 2013. Available from: `https://vimeo.com/71278954`

[131] Chikofsky, E.; Cross, J. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, volume 7, no. 1, Jan. 1990: pp. 13–17, ISSN 0740-7459, doi:10.1109/52.43044.

[132] Dit, B.; Revelle, M.; Gethers, M.; et al. Feature location in source code: a taxonomy and survey: FEATURE LOCATION IN SOURCE CODE: A TAXONOMY AND SURVEY. *Journal of Software: Evolution and Process*, volume 25, no. 1, Jan. 2013: pp. 53–95, ISSN 20477473, doi:10.1002/smr.567.

[133] Lau, K.-K.; Arshad, R. A Concise Classification of Reverse Engineering Approaches for Software Product Lines. In *International Journal On Advances in Software*, 2016.

[134] Ducasse, S.; Lanza, M.; Tichelaar, S. Moose: an extensible language-independent environment for reengineering object-oriented systems. In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, volume 4, 2000.

[135] Bergel, A. *Agile Visualization*. lulu.com, 2016, ISBN 978-1-365-31409-4.

[136] Girba, T. *The Moose Book*. online: Feenk, 2018. Available from: `http://www.themoosebook.org/`

[137] Mannaert, H.; Verelst, J. *Normalized Systems—Re-creating Information Technology Based on Laws for Software Evolvability*. Kermt, Belgium: Koppa, 2009.

[138] Mannaert, H.; Verelst, J.; De Bruyn, P. *Normalized systems theory: from foundations for evolvable software toward a general theory for evolvable design*. 2016, ISBN 978-90-77160-09-1.

[139] Herwig Mannaert; Jan Verelst; Kris Ven. Towards evolvable software architectures based on systems theoretic stability. Jan. 2011.

[140] Mannaert, H.; De Bruyn, P.; Verelst, J. Exploring entropy in software systems: towards a precise definition and design rules. In *Proceedings of the Seventh International Conference on Systems (ICONS)*, Saint Gilles, Reunion Island, 2012, pp. 93–99.

[141] Mannaert, H.; Verelst, J.; Ven, K. The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability. *Science of Computer Programming*, volume 76, no. 12, 2011: pp. 1210–1222, doi:10.1016/j.scico.2010.11.009.

[142] Van Nuffel, D.; Mannaert, H.; De Backer, C.; et al. Towards a deterministic business process modelling method based on normalized systems theory. *International Journal on Advances in Software*, volume 3, no. 1 & 2, 2010, ISSN 1942-2636.

[143] De Bruyn, P.; Mannaert, H. On the generalization of normalized systems concepts to the analysis and design of modules in systems and enterprise engineering. *International Journal on Advances in Systems and Measurements*, volume 5, no. 3& 4, 2012: pp. 216–232.

[144] De Bruyn, P.; Dierckx, G.; Mannaert, H. Aligning the normalized systems theorems with existing heuristic software engineering knowledge. In *Proceedings of the Seventh International Conference of Software Engineering Advances (ICSEA)*, Lisbon, Portugal, 2012, pp. 85–89.

[145] De Bruyn, P.; Huysmans, P.; Mannaert, H.; et al. Understanding Entropy Generation during the Execution of Business Process Instantiations: An Illustration from Cost Accounting. In *Advances in Enterprise Engineering VII*, *Lecture Notes in Business Information Processing*, volume 146, Springer Berlin Heidelberg, 2013, ISBN 978-3-642-38116-4, pp. 103–117.

[146] Verelst, J.; Silva, A. R.; Mannaert, H.; et al. Identifying Combinatorial Effects in Requirements Engineering. In *Advances in Enterprise Engineering VII*, Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg, May 2013, ISBN 978-3-642-38116-4, pp. 88–102, doi:10.1007/978-3-642-38117-1_7.

[147] McIlroy, M. D. Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*, 1968.

[148] Lidwell, W.; Holden, K.; Butler, J. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design*. Beverly, Mass: Rockport Publishers, second edition, revised and updated edition edition, Jan. 2010, ISBN 978-1-59253-587-3.

[149] Oorts, G.; Mannaert, H.; Bruyn, P. D.; et al. On the Evolvable and Traceable Design of (Under)graduate Education Programs. In *Advances in Enterprise Engineering X*, Lecture Notes in Business Information Processing, Springer, Cham, May 2016, ISBN 978-3-319-39566-1 978-3-319-39567-8, pp. 86–100, doi:10.1007/978-3-319-39567-8_6.

[150] Centre for Conceptual Modelling and Implementation. OntoUML.org. Available from: `https://ontouml.org`

[151] Bruyn, P. D.; Huysmans, P.; Mannaert, H. Tailoring an Analysis Approach for Developing Evolvable Software Systems: Experiences from Three Case Studies. In *2016 IEEE 18th Conference on Business Informatics (CBI)*, volume 01, Aug. 2016, pp. 208–217, doi:10.1109/CBI.2016.31.

[152] International Organization for Standardization. ISO/IEC/IEEE 24765:2010 - Systems and software engineering – Vocabulary. 2010. Available from: `https://www.iso.org/standard/50518.html`

[153] Kay, A. C. The early history of Smalltalk. *SIGPLAN Not.*, volume 28, no. 3, Mar. 1993: pp. 69–95, ISSN 0362-1340, doi:10.1145/155360.155364.

[154] Kay, A. Dr. Alan Kay on the Meaning of "Object-Oriented Programming". July 2003. Available from: `http://www.purl.org/stefan_ram/pub/doc_kay_oop_en`

[155] Goldberg, A.; Robson, D. *Smalltalk 80: The Language*. Addison-Wesley Professional, first edition, Jan. 1989, ISBN 0-201-13688-0.

[156] Meyer, B. *Eiffel the language Prentice Hall object-oriented series*. Prentice hall Upper Saddle River, NJ, USA, 1992.

[157] Eng, R. K. Syntax on a Post Card. Nov. 2017. Available from: `https://medium.com/@richardeng/syntax-on-a-post-card-cb6d85fabf88`

[158] Pharo. Available from: `http://www.pharo.org`

[159] Chis, A.; Gîrba, T.; Nierstrasz, O. The Moldable Inspector: a framework for domain-specific object inspection. In *Proceedings of International Workshop on Smalltalk Technologies (IWST 2014)*, 2014.

[160] Chiş, A.; Nierstrasz, O.; Syrel, A.; et al. The Moldable Inspector. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, Onward! 2015, New York, NY, USA: ACM, 2015, ISBN 978-1-4503-3688-8, pp. 44–60, doi:10.1145/2814228.2814234.

[161] Chiş, A.; Nierstrasz, O.; Gîrba, T. Towards moldable development tools. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, ACM, 2015, pp. 25–26.

[162] Chiş, A.; Gîrba, T.; Nierstrasz, O. The Moldable Debugger: A Framework for Developing Domain-Specific Debuggers. In *Software Language Engineering*, Lecture Notes in Computer Science, Springer, Cham, Sept. 2014, ISBN 978-3-319-11244-2 978-3-319-11245-9, pp. 102–121, doi:10.1007/978-3-319-11245-9_6.

[163] Pharo Consortium. Available from: `http://consortium.pharo.org/web`

[164] Hughes, J. Why Functional Programming Matters. *The Computer Journal*, volume 32, no. 2, Jan. 1989: pp. 98–107, ISSN 0010-4620, 1460-2067, doi:10.1093/comjnl/32.2.98.

[165] Hu, Z.; Hughes, J.; Wang, M. How functional programming mattered. *National Science Review*, volume 2, no. 3, Sept. 2015: pp. 349–370, ISSN 2095-5138, 2053-714X, doi:10.1093/nsr/nwv042.

[166] Halloway, S. *Programming Clojure*. Pragmatic Bookshelf, first edition, June 2009, ISBN 1-934356-33-6.

[167] VanderHart, L.; Sierra, S. *Practical Clojure*. Apress, first edition, June 2010, ISBN 1-4302-7231-7.

[168] Fogus, M.; Houser, C. *The joy of Clojure*. Manning Publications, 2011.

[169] Lipovača, M. *Learn You a Haskell for Great Good!: A Beginner's Guide*. No Starch Press, 2011, ISBN 978-1-59327-283-8.

[170] O'Sullivan, B.; Goerzen, J.; Stewart, D. B. *Real World Haskell: Code You Can Believe In*. "O'Reilly Media, Inc.", Nov. 2008, ISBN 978-0-596-55430-9.

[171] Bird, R. *Thinking Functionally with Haskell*. Cambridge University Press, Oct. 2014, ISBN 978-1-107-08720-0.

[172] Abelson, H.; Sussman, G. J.; Sussman, J. *Structure and Interpretation of Computer Programs - 2nd Edition*. The MIT Press, second edition edition edition, Sept. 1996, ISBN 978-0-262-51087-5.

[173] Atencio, L. *Functional Programming in JavaScript*. Manning, 2016.

[174] Gamma, E.; Helm, R.; Johnson, R.; et al. *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader)*. Pearson Education, Oct. 1994, ISBN 978-0-321-70069-8.

[175] Thatte, S. Quasi-static typing. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM, 1989, pp. 367–381.

[176] Abadi, M.; Cardelli, L.; Pierce, B.; et al. Dynamic typing in a statically typed language. *ACM transactions on programming languages and systems (TOPLAS)*, volume 13, no. 2, 1991: pp. 237–268.

[177] Meijer, E.; Drayton, P. Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. OOPSLA, 2004.

[178] Czaplicki, E.; Chong, S. Asynchronous functional reactive programming for GUIs. In *ACM SIGPLAN Notices*, volume 48, ACM, 2013, pp. 411–422.

[179] Pattnaik, K.; Mishra, B. S. P. Introduction to big data analysis. In *Techniques and Environments for Big Data Analysis*, Springer, 2016, pp. 1–20.

[180] Cooper, E.; Lindley, S.; Wadler, P.; et al. Links: Web programming without tiers. In *Formal Methods for Components and Objects*, Springer, 2007, pp. 266–296.

[181] Enterprise Architect. Available from: `http://sparxsystems.com/products/ea/`

[182] Visual Paradigm. Available from: `https://www.visual-paradigm.com`

[183] Smith, H.; Fingar, P. *Business process management: the third wave*, volume 1. Meghan-Kiffer Press Tampa, 2003.

[184] Aalst, W. M. P. v. d.; Hee, K. v. v. *Workflow Management: Models, Methods, and Systems*. Cambridge, Mass.: The MIT Press, Jan. 2004, ISBN 978-0-262-72046-5.

[185] Karagiannis, D.; Kühn, H. Metamodelling platforms. In *EC-Web*, volume 2455, 2002, p. 182.

[186] Uhnák, P.; Bommel, P. Facilitating the design of ABM and the code generation to promote participatory modelling. In *Environmental modelling and software for supporting a sustainable future*, 2016.

[187] Hunt, A.; Thomas, D. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, first edition, Oct. 1999, ISBN 0-201-61622-X.

Relevant Publications Authored and Co-Authored

[188] Jiří Vaníček; Martin Papík; Robert Pergl; et al. *Mathematical foundations of computer science*. Kernberg Publishing s.r.o., 2008, ISBN 978-80-87168-06-6.

[189] Michaël Verdonck; Frederik Gailly; Giancarlo Guizzardi; et al. Comparing traditional conceptual modeling with ontology-driven conceptual modeling: an empirical study. *Information Systems*, volume (submitted for review), 2018.

[190] Podloucký, M.; Pergl, R. Towards Formal Foundations for BORM ORD Validation and Simulation. In *Proceedings of the 16th International Conference on Enterprise Information Systems*, SCITEPRESS - Science and and Technology Publications, 2014, ISBN 978-989-758-027-7 978-989-758-028-4 978-989-758-029-1, pp. 315–322, doi:10.5220/0004897603150322.

[191] Robert Pergl; Lucie Houdová; Miloš Fetter. Marrow Donor Registry Simulator. 2017. Available from: `https://ccmi.fit.cvut.cz/mdr-simulator`

[192] Podloucký, M.; Pergl, R.; Kroha, P. Revisiting the BORM OR Diagram Composition Pattern. In *Enterprise and Organizational Modeling and Simulation*, *Lecture Notes in Business Information Processing*, volume 231, Stockholm: Springer, 2015, pp. 102–113, doi:10.1007/2F978-3-319-24626-0_8.

[193] Pícka, M.; Pergl, R. Gradual Modeling of Information System - Model of Method Expressed as Transitions Between Concepts. In *ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration, Paphos, Cyprus, May 23-27, 2006*, 2006, ISBN 978-972-8865-41-2, pp. 538–541. Available from: `5`

[194] Pergl, R.; Sales, T. P.; Rybola, Z. Instance-Level Modelling and Simulation Revisited. In *Enterprise and Organizational Modeling and Simulation*, Valencia,

Spain: Springer, June 2013, ISBN 978-3-642-41637-8, pp. 85 – 100, doi:10.1007/978-3-642-41638-5_6.

[195] Náplava, P.; Pergl, R. Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment. In *2015 IEEE 17th Conference on Business Informatics*, volume 2, July 2015, pp. 18–26, doi:10.1109/CBI.2015.12.

[196] Mráz, O.; Náplava, P.; Pergl, R.; et al. Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method. In *Advances in Enterprise Engineering XI: 7th Enterprise Engineering Working Conference, EEWC 2017, Antwerp, Belgium, May 8-12, 2017, Proceedings*, Cham: Springer International Publishing, 2017, ISBN 978-3-319-57955-9, pp. 85–98.

[197] Dudok, E.; Guerreiro, S.; Babkin, E.; et al. Enterprise Operational Analysis Using DEMO and the Enterprise Operating System. In *Advances in Enterprise Engineering IX*, number 211 in Lecture Notes in Business Information Processing, Springer International Publishing, June 2015, ISBN 978-3-319-19296-3 978-3-319-19297-0, pp. 3–18.

[198] Hornáčková, B.; Skotnica, M.; Pergl, R. Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering. In *Advances in Enterprise Engineering XII: 8th Enterprise Engineering Working Conference, EEWC 2018, Luxembourg, Luxembourg, Proceedings*, Cham: Springer International Publishing, 2018.

[199] Struska, Z.; Pergl, R. BORM-points: Introduction and Results of Practical Testing. In *Lecture Notes in Business Information Processing*, volume 24, Milan, Italy: Springer Berlin Heidelberg, 2009, pp. 590–599.

[200] Podloucký, M.; Pergl, R. The Prefix Machine – a Formal Foundation for the BORM OR Diagrams Validation and Simulation. In *Enterprise and Organizational Modeling and Simulation*, volume 191, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ISBN 978-3-662-44859-5 978-3-662-44860-1, pp. 113–131.

[201] Petr Šplíchal; Robert Pergl; Marek Pícka. BORM model transformation. *Systémová integrace*, volume 18, no. 2, 2011, ISSN 1210-9479.

[202] Pergl, R.; Sales, T. P.; Rybola, Z. Towards OntoUML for Software Engineering: From Domain Ontology to Implementation Model. In *Proceedings of MEDI 2013*, volume 3rd, Amantea, Italy: Springer, Sept. 2013, ISBN 978-3-642-41365-0, pp. 249–263, doi:10.1007/978-3-642-41366-7.

[203] Rybola, Z.; Pergl, R. Towards OntoUML for Software Engineering: Introduction to the Transformation of OntoUML into Relational Databases. In *Enterprise and Organizational Modeling and Simulation*, LNBIP, Ljubljana, Slovenia: Springer, June 2016, ISBN 978-3-319-49453-1, doi:10.1007/978-3-319-49454-8_5.

[204] Rybola, Z.; Pergl, R. Towards OntoUML for Software Engineering: Transformation of Rigid Sortal Types into Relational Databases. In *Proceedings of {FedCSIS} 2016, ACSIS*, volume 8, Gdańsk, Poland: IEEE, Sept. 2016, ISBN 978-83-60810-90-3, pp. 1581–1591, doi:10.15439/2016F250.

[205] Rybola, Z.; Pergl, R. Towards OntoUML for Software Engineering: Transformation of Kinds and Subkinds into Relational Databases. *Computer Science and Information Systems*, volume 14, no. 3, 2017: pp. 913–937, ISSN 1820-0214, doi: 10.2298/CSIS170109035R.

[206] Rybola, Z.; Pergl, R. Towards OntoUML for Software Engineering: Transformation of Anti-Rigid Sortal Types into Relational Databases. In *Model and Data Engineering*, LNCS, Aguadulce, Almería, Spain: Springer, Sept. 2016, ISBN 978-3-319-45546-4, pp. 1–15, doi:10.1007/978-3-319-45547-1_1.

[207] Zdeněk Rybola; Robert Pergl. Transformation of Kinds and Subkinds into Relational Databases: A Running Example. Technical Report TR-FIT-2017, Czech Technical University in Prague, 2017.

[208] Rybola, Z.; Pergl, R. Towards OntoUML for Software Engineering: Optimizing Kinds and Subkinds Transformed into Relational Databases. In *Enterprise and Organizational Modeling and Simulation*, Tallinn, Estonia, Nov. 2018.

[209] Dvořák, O.; Pergl, R.; Kroha, P. Confirmation Engine Design Based on PSI Theory. In *17th IEEE Conference on Business Informatics, Workshop on Cross-organizational and Crosscompany BPM (XOC-BPM) Lisbon*, 2015.

[210] Skotnica, M.; Kervel, S. J. H. v.; Pergl, R. Towards the Ontological Foundations for the Software Executable DEMO Action and Fact Models. In *Advances in Enterprise Engineering X*, Funchal, Madeira: Springer International Publishing, May 2016, pp. 151–165, doi:10.1007/978-3-319-39567-8_10.

[211] Skotnica, M.; van Kervel, S. J. H.; Pergl, R. A DEMO Machine - A Formal Foundation for Execution of DEMO Models. In *Advances in Enterprise Engineering XI: 7th Enterprise Engineering Working Conference, EEWC 2017, Antwerp, Belgium, May 8-12, 2017, Proceedings*, Cham: Springer International Publishing, 2017, ISBN 978-3-319-57955-9, pp. 18–32.

[212] Uhnák, P.; Pergl, R. Ad-hoc Runtime Object Structure Visualizations with MetaLinks. In *Proceedings of the 12th Edition of the International Workshop on Smalltalk Technologies*, IWST '17, New York, NY, USA: ACM, 2017, ISBN 978-1-4503-5554-4, pp. 7:1–7:10, doi:10.1145/3139903.3139912.

[213] Dvořák, O.; Pergl, R.; Kroha, P. Affordance-driven Software Assembling. In *Advances in Enterprise Engineering XII: 8th Enterprise Engineering Working Conference, EEWC 2018, Luxembourg, Luxembourg, Proceedings*, Cham: Springer International Publishing, 2018.

[214] Dvořák, O.; Pergl, R.; Kroha, P. Tackling the Flexibility-Usability Trade-off in Component-Based Software Development. In *Recent Advances in Information Systems and Technologies*, Advances in Intelligent Systems and Computing, Springer, Cham, Apr. 2017, ISBN 978-3-319-56534-7 978-3-319-56535-4, pp. 861–871, doi: 10.1007/978-3-319-56535-4_84.

[215] Suchánek, M.; Pergl, R. Evolvable Documents – an Initial Conceptualization. Barcelona, ES: IARIA, Feb. 2018, ISBN 978-1-61208-612-5, pp. 39 – 44.

[216] Nedvedova, K.; Pergl, R. Information Support Systems for Cultural Heritage Protection Against Flooding. In *25th International Cipa Symposium 2015*, volume 40-5, Gottingen: Copernicus Gesellschaft Mbh, 2015, pp. 343–346.

[217] Bliznič enko, J.; Papoulias, N.; Pergl, R.; et al. Towards Modularity in Live Visual Modeling: A Case Study with OpenPonk and Kendrick. In *Proceedings of the 12th Edition of the International Workshop on Smalltalk Technologies*, IWST '17, New York, NY, USA: ACM, 2017, ISBN 978-1-4503-5554-4, pp. 3:1–3:10, doi:10.1145/3139903.3139908.

[218] Janeček, L.; Pergl, R. Analysing Functional Paradigm Concepts: The JavaScript Case. In *Recent Advances in Information Systems and Technologies*, Advances in Intelligent Systems and Computing, Springer, Cham, Apr. 2017, ISBN 978-3-319-56534-7 978-3-319-56535-4, pp. 882–891, doi:10.1007/978-3-319-56535-4_86.

[219] Pergl, R. Modelling and prototyping of business applications based on multilevel domain-specific language. *Lecture Notes in Business Information Processing*, volume 88 LNBIP, 2011: pp. 173–191.

[220] Suchánek, M.; Pergl, R.; Hooft, R.; et al. Data Stewardship Wizard (poster). Berlin, Germany, June 2018, doi:10.7490/f1000research.1115594.1. Available from: `https://www.elixir-europe.org/events/elixir-all-hands-2018`

[221] Pergl, R. Supporting enterprise IS modelling using ontological analysis. *Lecture Notes in Business Information Processing*, volume 88 LNBIP, 2011: pp. 130–144.

[222] Pergl, R.; Tůma, J. OpenCASE - A tool for ontology-centred conceptual modelling. *Lecture Notes in Business Information Processing*, volume 112 LNBIP, 2012: pp. 511–518.

[223] Merunka, V.; Pergl, R.; Tůma, J. BORM-II and UML as Accessibility Process in Knowledge and Business Modelling. In *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*, number 312 in Lecture Notes in Electrical Engineering, Springer International Publishing, 2015, ISBN 978-3-319-06763-6 978-3-319-06764-3, pp. 1–6.

[224] Uhnák, P.; Pergl, R. The OpenPonk Modeling Platform. In *Proceedings of the 11th Edition of the International Workshop on Smalltalk Technologies*, IWST'16, New York, NY, USA: ACM, 2016, ISBN 978-1-4503-4524-8, pp. 14:1–14:11, doi:10.1145/2991041.2991055.

Relevant Supervised Bachelor's and Master's Theses

[225] Král, O. *Ontological Analysis of ICT Project Change Management*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2018. Available from: `https://dspace.cvut.cz/handle/10467/77477`

[226] Svoboda, J. *OpenPonk: an Implementation of a Parser and Interpreter of OCL*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2018. Available from: `https://dspace.cvut.cz/handle/10467/76663`

[227] Kovář, M. *Survey of Conceptual Modelling Utilisation in Companies*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2015. Available from: `https://dspace.cvut.cz/handle/10467/63185`

[228] Moravec, L. *Ontological Analysis of the CTU Data Warehouse*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, May 2016. Available from: `https://dspace.cvut.cz/handle/10467/65120`

[229] Žmolík, J. *Optimization of Manufacturing Process in Enterprise*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2015. Available from: `https://dspace.cvut.cz/handle/10467/63004`

[230] Žďára, A. *DEMO Diagrams Visualisation for Managers*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2015. Available from: `https://dspace.cvut.cz/handle/10467/63001`

[231] Larionova, V. *Discrete Simulation for the BORM Method in the OpenCABE Tool*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2015. Available from: `https://dspace.cvut.cz/handle/10467/63143`

[232] Vološin, M. *Transformation of OntoUML into Smalltalk*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2014.

[233] Homola, D. *Model-Driven Engineering Approach for OntoUML*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016. Available from: https://dspace.cvut.cz/handle/10467/65132

[234] Skotnica, M. *Implementation of a module supporting the AM model in the Formetis DEMO Processor*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2014. Available from: https://dspace.cvut.cz/handle/10467/25079

[235] Skotnica, M. *Towards the Foundations of Fact and Rules Ontology for Discrete Systems*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016.

[236] Suchánek, M. *Conceptual Modelling Support for the Haskell Programming Language*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, May 2017. Available from: https://dspace.cvut.cz/handle/10467/69146

[237] Larionova, V. *Reverse Engineering of Legacy Software Code for Normalized Systems Exanders*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2018.

[238] Kolařík, V. *Applying OntoUML for stuctural definitons of normalized systems expanders*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2014. Available from: https://dspace.cvut.cz/handle/10467/24458

[239] Uhnák, P. *Developing Normalized Systems Conceptual Modeler*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2018. Available from: https://dspace.cvut.cz/handle/10467/76365

[240] Balda, M. *A System for Matching Offers and Inquiries Based on the Pure Object-Oriented Paradigm*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2015. Available from: https://dspace.cvut.cz/handle/10467/63025

[241] Blizničenko, J. *Live Visualization of Epidemiological Models*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2017. Available from: https://dspace.cvut.cz/handle/10467/73230

[242] Tvrz, T. *The Groovy language as an alternative for business applications development*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2014. Available from: https://dspace.cvut.cz/handle/10467/25080

[243] Ševčík, J. *Web application using functional approach*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2014. Available from: https://dspace.cvut.cz/handle/10467/24475

[244] Altman, O. *Study of Utilising ClojureScript Technology in a Developer Company*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2015. Available from: https://dspace.cvut.cz/handle/10467/63160

[245] Luxemburk, J. *Functional Programming for Web Frontend*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, May 2017. Available from: https://dspace.cvut.cz/handle/10467/69598

[246] Podloucký, M. *Automated GUI Generation for functional data structures*. Master's thesis, Charles University, 2012. Available from: https://is.cuni.cz/webapps/zzp/detail/107904/

[247] Knaisl, V. *Migration Tool for Data Stewardship Knowledge Model*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Jan. 2018. Available from: https://dspace.cvut.cz/handle/10467/73973

[248] Slifka, J. *Data Stewardship Portal: Webový Frontend*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Jan. 2018. Available from: https://dspace.cvut.cz/handle/10467/73980

[249] Balda, M. *Portal for BORM processes optimization*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2013. Available from: https://dspace.cvut.cz/handle/10467/17752

[250] Zyková, A. *BORM process machine*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2014. Available from: https://dspace.cvut.cz/handle/10467/24435

[251] Lanský, R. *Messaging and Task Management Application Based on the PSI Theory*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Feb. 2017. Available from: https://dspace.cvut.cz/handle/10467/69103

[252] Uhnák, P. *Layouting of Diagrams in the DynaCASE Tool*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016. Available from: https://dspace.cvut.cz/handle/10467/63194

[253] Bambas, J. *Validation of Process Diagrams in BORM Method*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2015. Available from: https://dspace.cvut.cz/handle/10467/62916

[254] Valášek, J. *Generating reports from BORM process diagrams*. bachelor's thesis, Czech Technical University in Prague, June 2015. Available from: https://dspace.cvut.cz/handle/10467/63173

[255] Blizničenko, J. *Simulation and Visualisation Support for the DynaCASE Tool*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2015. Available from: https://dspace.cvut.cz/handle/10467/63136

[256] Turoň, J. *Enhancing the DynaCASE Platform for Enterprise Engineering*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016.

[257] Anisimov, B. *Support of BPMN Standard on OpenPonk Platform*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2018. Available from: `https://dspace.cvut.cz/handle/10467/76633`

[258] Vološin, M. *OntoUML Models Instance Visualisation*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, May 2016. Available from: `https://dspace.cvut.cz/handle/10467/65114`

Relevant Reviewed Bachelor's and Master's Theses

[259] Hakala, M. *Concepts of IT Architecture Building*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2015.

[260] Kozlov, A. *Specification of Cloud Computing Resources Based on Ontological Description*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2018. Available from: `https://dspace.cvut.cz/handle/10467/76423`

[261] Homola, D. *Comparison of a data model analysis using OntoUML and UML*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2014. Available from: `https://dspace.cvut.cz/handle/10467/24536`

[262] Cimpl, L. *Ways of Business Process Modelling Notations Comparison*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016. Available from: `https://dspace.cvut.cz/handle/10467/62774`

[263] Posoldová, A. *DEMO to BPMN Transformation Ways*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, May 2017. Available from: `https://dspace.cvut.cz/handle/10467/70169`

[264] Heller, S. *Usage of DEMO Methods for BPMN Models Creation*. Master's thesis, Czech Technical University in Prague. Computing and Information Centre., 2016. Available from: `http://hdl.handle.net/10467/62776`

[265] Jirovský, V. *Conceptual Analsis of of Data Domains of Study and Teaching Quality Evaluation Agendas with Respect to Data Cleanness*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2015.

[266] Mráz, O. *DEMO Principles Possibilities for Improving BPMN Models Quality*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016.

[267] Nymsa, P. *Mobile Enterprise Architecture Process Analytic Tool Based on the DEMO Methodology*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2018. Available from: `https://dspace.cvut.cz/handle/10467/77482`

[268] Janeček, L. *Study of choice of a database platform for realization of normalized software systems*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2015. Available from: `https://dspace.cvut.cz/handle/10467/62995`

[269] Fibichr, J. *Leveraging Toolkits in BPM Application Development*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2013. Available from: `https://dspace.cvut.cz/handle/10467/16304`

[270] Jelínková, K. *Comparison of Process Application Operation in IBM BPM and Activiti BPMS*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2014. Available from: `https://dspace.cvut.cz/handle/10467/24516`

[271] Šimon, V. *Comparison of process application development and operation in IBM BPM and Open Source BPMS*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2014. Available from: `https://dspace.cvut.cz/handle/10467/24504`

[272] Lanský, R. *Components for the "SZZ" Process System*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2013. Available from: `https://dspace.cvut.cz/handle/10467/17853`

[273] Maxa, O. *Implementation of process analysis of BORM OR diagrams*. bachelor's thesis, Charles University, 2015. Available from: `https://is.cuni.cz/webapps/zzp/detail/143406/`

[274] Buša, R. *Designing WYSIWYG Web Forms*. bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, June 2018. Available from: `https://dspace.cvut.cz/handle/10467/77488`