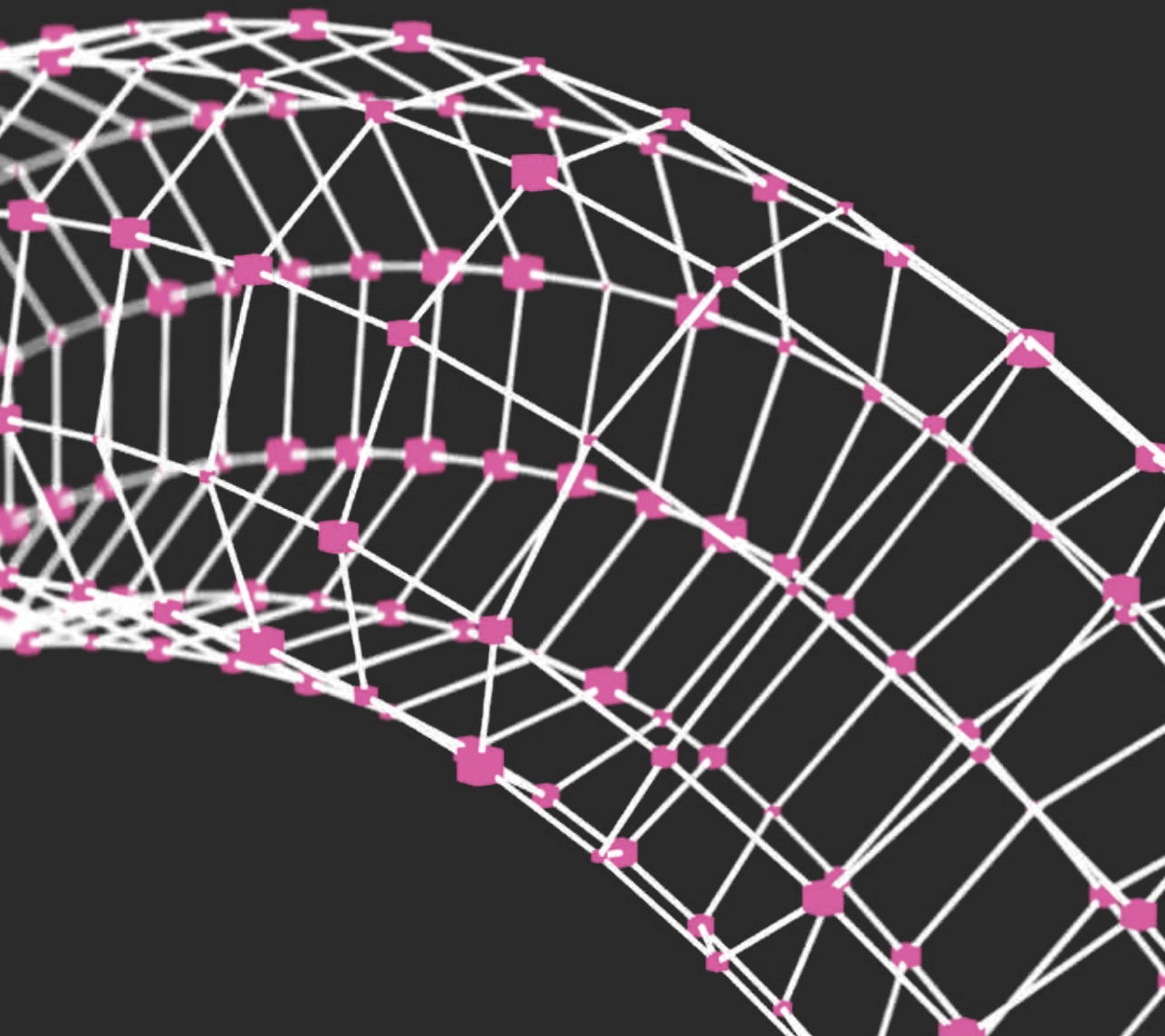
On the design of enterprise ontology-driven software development

On the Design of Enterprise Ontology-Driven Software Development

Marien R. Krouwel



Over het ontwerp van organisatieontologiegestuurde softwareontwikkeling

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Maastricht,
op gezag van de Rector Magnificus, Prof. dr. Pamela Habibović
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op vrijdag 3 november 2023 om 13:00 uur

door

Marien Rolin Krouwel
geboren op 12 april 1986
te Utrecht

Promotoren:
Prof. dr. A.F. Harmsen
Prof. dr. H.A. Proper, Technische Universiteit Wenen, Wenen, Oostenrijk

Co-promotor:
Prof. dr. M. Op 't Land, Antwerp Management School, Antwerpen, België

Manuscriptcommissie:
Prof. dr. A. Brüggen (voorzitter)
Prof. dr. M.G.J. van den Brand, Technische Universiteit Eindhoven, Eindhoven, Nederland
Prof. dr. H.F.D. Hassink
Ing. R. Pergl, Technische Universiteit Tsjechië, Praag, Tsjechië

On the Design of Enterprise Ontology-Driven Software Development

DISSERTATION

to obtain the degree of Doctor at the Maastricht University,
on the authority of the Rector Magnificus, Prof. dr. Pamela Habibović
in accordance with the decision of the Board of Deans,
to be defended in public
on Friday, November 3, 2023, at 13:00 hours

by

Marien Rolin Krouwel
born on April 12, 1986
in Utrecht (the Netherlands)

Supervisors:
Prof. dr. A.F. Harmsen
Prof. dr. H.A. Proper, Technische Universität Wien, Vienna, Austria

Co-supervisor:
Prof. dr. M. Op 't Land, Antwerp Management School, Antwerp, Belgium

Assessment Committee:
Prof. dr. A. Brüggen (chair)
Prof. dr. M.G.J. van den Brand, Eindhoven University of Technology, Eindhoven, the Netherlands
Prof. dr. H.F.D. Hassink
Ing. R. Pergl, Czech Technical University, Prague, Czech Republic

Arundo et Olea seu Quercus

Quercus, et adversans illi contendit Arundo,
Utraque praestantem se magis esse refert.
Tune quamvis Quercu quod mobilis esset ad auram
Obiciente, lubens Canna modesta tacet.
Tempore sed parvo post diruta turbine Quercus
Dicitur, et ramis fracta fuisse suis.
At varie flectens se ventis cessit Arundo,
Hac ea non frangi mobilitate potest.
Laudandi potius sunt, qui concedere norunt,
Quam prae se fortes qui superare parant.
Noveris esse Deo, convellat ut ardua, morem,
Stare diu quo non summa premente valent.

From *Phryx Aesopus Habitu Poetico* by Hieronymus Osius, 1574[1]

[1] A simplified Latin version by Laura Gibbs is available at https://archive.org/details/gibbs-laura-mille-fabulae-et-una.-1001-aesops-fables-in-latin.

The Oak and the Reeds

A Giant Oak stood near a brook in which grew some slender Reeds. When the wind blew, the great Oak stood proudly upright with its hundred arms uplifted to the sky. But the Reeds bowed low in the wind and sang a sad and mournful song. "You have reason to complain," said the Oak. "The slightest breeze that ruffles the surface of the water makes you bow your heads, while I, the mighty Oak, stand upright and firm before the howling tempest." "Do not worry about us," replied the Reeds. "The winds do not harm us. We bow before them and so we do not break. You, in all your pride and strength, have so far resisted their blows. But the end is coming." As the Reeds spoke a great hurricane rushed out of the north. The Oak stood proudly and fought against the storm, while the yielding Reeds bowed low. The wind redoubled in fury, and all at once the great tree fell, torn up by the roots, and lay among the pitying Reeds.

From *The Æsop for Children: with Pictures by Milo Winter*, 1919[2]

[2]Available as ebook by Project Gutenberg EBook at https://www.gutenberg.org/ebooks/19994 and as interactive book by Library of Congress at https://read.gov/aesop/index.html.

"It is better to concede when it is foolish to resist,
than to resist stubbornly and be destroyed."

Based on a story by Aesop (∼600B.C.)

Preface

The story and the poem on the previous pages originate from a collection of fables credited to Aesop – Aesopus or Aisopos –, a Greek storyteller from around 600 BC. These fables are known as Aesop's Fables or 'the Aesopica'. The moral of the story is that those who adapt will emerge unharmed, as summarized in the quote above.

While this moral holds for life in general, I apply it specifically to my work with enterprises and the software that supports these enterprises. Only when an enterprise can adapt to its changing environment, it will be able to survive. In a world where every enterprise uses software, and where more and more enterprises are becoming a technology company,[3], it is of the utmost importance that enterprise software is adaptable as well. Adaptable software is not new, but configuring standard packages and creating customized software is still a time-consuming task. There is a need for software that is *easily* and *quickly adaptable* to typical changes in the enterprise.

Back to the story: an oak will never become reed, or vice versa. While both may adapt in response to a changing environment, even into a new species, there is a stable core of DNA sequences that differentiates the oak from reed. A similar reasoning holds for enterprises: a bank will never (suddenly) become a pizzeria.

With DEMO Enterprise Ontology I found a way to discover the stable core of an enterprise. Subsequently, I devised a framework to explicitly design adaptability around this stable core. Model-driven Software Development brought me a way to quickly convert these enterprise models into working software. Together, these concepts provide the ingredients to create enterprise software that is designed to quickly adapt to changing enterprise needs. This thesis provides the foundation for a new method, called *Enterprise Ontology-Driven Software Development*, to create enterprise software that is adaptable around a stable core.

About the cover

The structure shows a network of related (organizational and software) components. Although the structure is built with rigid components, the structure as a whole is very flexible so that it can adapt to a changing environment.

[3] See, e.g., [217], https://www.ing.com/Newsroom/News/We-want-to-be-a-tech-company-with-a-banking-license-Ralph-Hamers.htm, https://www.satellitetoday.com/innovation/2019/02/26/microsoft-ceo-every-company-is-now-a-software-company/ and https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/every-company-is-a-software-company-six-must-dos-to-succeed.

"It is good to have an end to journey towards; but it is the journey that matters, in the end."

Ursula K. Le Guin, in "The Left Hand of Darkness" (1969)

Acknowledgements

Some say it's the journey that matters, not the destination or end goal.[4] And while the journey has been great, I am very happy that my PhD journey now comes to an end. It has been a bumpy and long road, that brought me to many places in which I met a lot of people.

In 2013, I took my first steps in setting up my research at University of Antwerp. During my time in Antwerp, I got to know my fellow PhD students: Dieter, Els, Jos, Mark, Peter, and Philippe. I want to thank you all for the inspiring conversations, with or without pizza after the EDSM course. I want to thank Jan Verelst and Herwig Mannaert for their support in starting my PhD. Additionally, I want to thank Lourens van der Weerd, as he was the one that made it possible to combine my PhD with my full-time job at Capgemini.

During my time at Capgemini I have met many people with interests that are related to or at the core of my research; in reverse order of appearance: Alcedo Coenen (ABN AMRO), Steven Hanekroot (Dutch Police), Steven Gort (ICTU), Diederik Dulfer c.s. (Dutch Tax Authority), and Edward van Dipten (Dutch Ministry of Defense). I thank you all, including the ones I forgot to mention, for providing me the opportunity to combine research and consultancy. I also want to thank the people that I have met in a DEMO or Enterprise Design Course, where we often had lively discussions on the theories behind DEMO and the practical challenges in implementing a DEMO model. I also am thankful for the opportunities I had in talking to representatives and board members of different technology vendors, and Mendix and USoft in particular, as those conversations helped to shape my ideas and convert them into a business opportunity.

During my research, I have worked together with several (MSc) students, among which were Sam, Sandra, Lotte, Jermaine, and Jelle. I am thankful for the contributions you made to my research, as well as the challenging discussions we had. I am also thankful to the organizers of the conferences I attended, and especially David Aveiro (EEWC, Madeira), Antonio Albani (EEWC), Isabella Ramos (MCIS, Guimarães), Robert Pergl (EEWC), and Kurt Sandkuhl (PoEM, London), as these conferences allowed me to improve my research and collaborate with other researchers.

This research wouldn't exist if I hadn't seen the field of Enterprise Engineering. I am very grateful to Jan Dietz as he introduced me into the DEMO way of thinking, and for his support and feedback all along the way. On the topic of

[4] In a way, this whole thesis describes a journey, from enterprise models to working software; it is up to the reader to make this journey and arrive in a destination of adaptable enterprise software.

DEMO I also want to thank Ingrid Theuwissen, Peter Kuipers, and Tine de Mik for providing an opportunity to really dive into the EE-theories in an effort to explain them as easy as possible.[5]

Referring to the story of the oak and the reeds: sometimes I was foolish, sometimes I was stubborn, and it was the input from my supervisors, and my ability to deal with it, that improved this research significantly. I am very thankful to Erik Proper for transferring my research to Nijmegen and then to Maastricht, and for the assistance in finding a focus. I am also very thankful to Frank Harmsen for his critical view on the several choices I made in my research. I want to thank the assessment committee for their feedback as it contributed to the quality of this research.

I am very grateful to Hans Mulder as he was one of the stable factors during my PhD journey. Hans was there whenever I was struggling to progress, either he listened or provided guidance towards the next step in my research. Finally, I am very thankful to Martin Op 't Land, who introduced me into Capgemini and into the world of (practical) research. And while it was hard to combine a full-time job and academic research, Martin always showed me ways to find shared interests and he never stopped encouraging me to find the right balance between the two.

I want to thank my family, friends and paranymphs for their support: Florian, Violette, Dorien, Max, Rogier, Wiemer, Wouter, and ... (insert your name here in case I forgot to mention you). I especially want to thank my father, Dinant, for his interest in the topic, helping me find the correct references, and for proofreading my entire thesis. I am thankful to my mother, Annemarie, who has always challenged me to get the most out of myself. Although you cannot be here to see me receiving the highest academic grade, I know you are proud of me.

Last but far from least, I want to thank my wife, Janneke. You chose the better path in finishing your PhD before we got married. I chose the difficult path and tried to finish my PhD with two young kids running around. I am very grateful for your endless support, and even more grateful for the two possible future doctors, Sem and Lize, we raise.

Marien, September 18, 2023, Utrecht

[5]The result can be downloaded from https://ee-institute.org/demo/werken-met-demo/.

Abstract

Due to factors such as hyper-competition, increasing expectations from customers, regulatory changes, and technological advancements, the conditions in which enterprises need to thrive become increasingly turbulent. As a result, enterprise agility, or flexibility, becomes an increasingly important determinant for enterprise success. As enterprises rely more and more on software, the need for a software development method aimed at supporting the end users, adaptability, speed and traceability is almost evident.

The dream that drives research, is to create such a method, but many uncertainties are recognized in order to do so, including its feasibility. As methods, or elements thereof, often arise from structuring or generalizing procedures or approaches that are being applied in practice, a practice-driven Action Design Research approach is adopted.

Model-driven Software Development is an approach towards software development that relies on model transformations, supporting speed and traceability. The ontological model of an enterprise, as defined by the Design and Engineering Methodology for Organizations (DEMO), has shown to be useful in designing the stable parts of an enterprise and its supporting software. As part of this research, the Enterprise Implementation Framework is created that can be used to explicitly design, but linked to or as an extension of DEMO models, the more flexible implementation of an enterprise and its supporting software. Together they form the input for the Enterprise Model-driven Software Development approach as adopted in this research. Although there are existing method elements that could support such a method, these elements also have their limitations.

In four exploratory case studies it has been shown that creating a transformation from the input models to different target technologies is technically and procedurally feasible. Several possible method elements have been created for the automated and thus traceable and high-speed transformation of DEMO models into working software that is adaptable in consciously chosen enterprise implementation categories. With these elements and the results of the exploratory case studies, the foundation is laid to further create a method for the automatic creation of software from enterprise ontological models.

Summary

Situation and Goal

Due to factors such as hyper-competition, increasing expectations from customers, regulatory changes, and technological advancements, the conditions in which enterprises need to thrive become increasingly turbulent. As a result, the ability to change with an ever decreasing time-to-market, often referred to as 'agility', becomes an important determinant for the success of enterprises. As an enterprise and its supporting software can hardly be separated anymore, enterprise agility is to a large extent dependent on software agility.

Agility currently is mainly achieved by adopting processes that support flexibility with regard to planning and execution. In contrast to, e.g., a waterfall approach, agile approaches such as SCRUM and DevOps prescribe multidisciplinary teams that create new software versions in short iterations or even continuously. These approaches however say nothing about the structures in enterprises or its supporting software that should be adhered to in order to ensure quick adaptation is possible at all. It has been shown that changing software over time becomes more and more costly and indeed hampers enterprise agility.

The dream that drives this research is to have a structured approach towards software development for enterprises that supports the quick and continuous creation and adaptation of high quality software solutions to support enterprise agility. From the generic need for software quality, specific needs are identified, viz., that the software supports the end users, is evolvable, is created (or adapted) with little effort and in little time, and that the step from requirements to software constructs is traceable.

Partial answers are found in having a structured approach, i.e., a method. Methods for software development date back to the 1970s and mostly rely on the use of models. This is also known as Model-based Engineering (MBE), of which Model-driven Architecture (MDA), model-as-code and low code/no code are recent implementations. The reason that MBE and structured approaches gain new attention is the advancements in the availability of technologies as well as in (enterprise) modeling techniques. It is therefore that the challenge of this research is to work towards the creation of a new method for software development that answers the needs and supports enterprise agility. In order to meet this research goal, seven research questions have been formulated.

Approach

The dream that drives this research, is to have a method, including supporting tools, to create software from enterprise models that addresses the needs, can be applied repeatedly, and is adaptable to specific situations. In order to create such a method and supporting tools, clear specifications are needed. However, there are several uncertainties in order to create such a method, including what models to choose and whether such an approach is technically feasible.

It is because of these uncertainties that this research is considered part of the fuzzy front-end of creating a complete method. This fuzzy front-end is a necessary stage in which a problem is explored, guided by a vague idea of the solution, and assumptions are being examined, resulting in a minimum specification as well as an initial version of elements that could be part of the final solution. In this research the focus is on reducing these uncertainties, while aiming to find some initial method elements that could be part of such a method.

As methods often result from structuring or generalizing procedures or approaches that are being applied in practice, a practice-driven research approach is adopted. In order to deal with the uncertainties and answer the research questions, the Action Design Research (ADR) approach is adopted, combining Design Science Research with Action Research. ADR defines four stages of research that helps researchers to both make scientific contributions and to assist in solving current and anticipated problems of practitioners.

The research questions are answered initially through literature study, and validated or extended by practical research. The practical research consists of four exploratory case studies, aimed to explore the creation of a single method element. By combining theory and practice and by having multiple exploratory case studies, both rigor and relevance are added to this research.

Results and Benefits

The Model-driven Software Development (MDSD) approach is adopted as enabler for speed and traceability in the software development process. MDSD relies on model transformations for which both input models and target technologies are selected. From the possible enterprise modeling techniques, Design and Engineering Methodology for Organizations (DEMO) is chosen that provide the ontological enterprise models as the starting point for the MDSD approach. As DEMO models alone do not provide enough information to fully create working software, the Enterprise Implementation Framework has been developed. This framework uses Organization Implementation Variables (OIVs) to capture (additional) enterprise implementation design decisions, and can be used to consciously decide about the required flexibility on the enterprise level, that needs to be supported by the software. Together, DEMO and OIVs provide an answer to the other needs, i.e., completeness of user requirements and adaptability. An argument against this approach is that it moves complexity from the code to the (enterprise) models. As most complexity in software actually comes from the enterprise implementation, moving the complexity to these models is considered the only right approach.

Creating a (situational) method, mainly relies on the availability of so-called method fragments. Fragments can be categorized in three axes: perspective, abstraction and granularity. Existing literature is reviewed on the existence of method fragments that start from DEMO models and ends in some software implementation. While there are some usable fragments, none of these support all concepts from DEMO, most existing fragments ignore the step from DEMO to implementation model, and that there are almost no technical fragments to support the desired method.

For the four exploratory case studies four (modern) target technologies have been selected that all address multiple identified needs: microservices, mockups, Normalized Systems, and low code. The exploratory studies have in common that a mapping from the input, i.e., DEMO models and enterprise implementation (in terms of OIVs), to the target technology has been devised and evaluated in practice, sometimes on multiple enterprises (or enterprise models). These mappings, that are either automatable or automated during the exploratory case study, can be considered a method fragment, possibly to be assembled into an overall method. The exploratory case studies show that creating software from the chosen input models is indeed technically feasible, and that it is possible to do this in a structured (and automatable) way.

Put together, the foundation has been laid to create the desired method that addresses the needs and supports enterprise agility. The requirements are detailed and several fragments have been identified or created that can later be used to compose such a method. Such a method could be useful for practitioners as it may reduce software development costs and efforts, relies less on technically skilled people – so-called citizen development – and may improve project success rate.

Limitations and Future Research

The exploratory case studies were performed mainly in the Netherlands and mainly at public or semi-public organizations. Although there is no reason to believe that the results are not applicable to other enterprises, further validation is needed.

Moreover, most of the model transformations in the exploratory case studies involved manual steps. A reason simply is the unavailability of proper modeling tools for DEMO. In order to apply the algorithms in a disciplined way, it is necessary to further automate the mappings, and thus provide tool support to create the input models.

Embedding the suggested approach into existing approaches seems key to improve its adoption. Advantages are expected by introducing LLM and (generative) AI into the method. However, combining approach also means that they should be comparable, in their underlying theoretical background or 'world view', but also in the results they produce. Further research is needed in order to compare different approaches for software development from enterprise models.

Finally, adaptable software is just one factor in achieving enterprise agility. Other research shows that it is possible to apply principles to the enterprise level

that should enable enterprise agility. An open question is whether enterprise constructs can be created, perhaps based on one or more DEMO concepts, that inherently adhere to these principles. This also implies the need for an objective way to measure enterprise agility in order to be able to compare different enterprise constructs. Although the foundation was laid for creating enterprise software from enterprise models that intrinsically supports enterprise agility, there is still much to do in order to achieve true enterprise agility.

Nederlandse samenvatting

Aanleiding en doel

Organisaties[6] bevinden zich in toenemende mate in onstuimige omstandigheden, veroorzaakt door o.a. sterke concurrentie, toenemende verwachtingen van klanten, wijzigende wet- en regelgeving en technologische vooruitgang. Als gevolg hiervan wordt het vermogen om steeds sneller te kunnen veranderen, ook wel bekend als wendbaarheid of 'agility', steeds belangrijker voor het succes van organisaties. Omdat organisaties steeds meer afhankelijk zijn geworden van IT, is de wendbaarheid van organisaties steeds meer afhankelijk van de wendbaarheid van het IT landschap.

Wendbaarheid tracht men momenteel vooral te bereiken door processen in te richten die de flexibiliteit in de operatie en planning ondersteunen. Bekende methoden hiervoor zijn SCRUM en DevOps, die, in tegenstelling tot watervalaanpakken, voorschrijven dat multidisciplinaire teams in korte iteraties of zelfs continu software (door)ontwikkelen. Hoewel de voordelen van deze methoden duidelijk zijn, zeggen de methoden niets over hoe de software zelf in de constructie opgebouwd dient te worden, zodat aanpassingen überhaupt snel kunnen worden gemaakt. Wat niet helpt is de wet van Lehmann die zegt dat het wijzigen van software alleen maar duurder wordt en meer tijd gaat kosten, iets wat uiteindelijk ten koste gaat van de wendbaarheid van de software èn (dus) van de organisatie.

De droom achter dit onderzoek is om met behulp van een gestructureerde aanpak snel en continu nieuwe (versies van) software van hoge kwaliteit te kunnen neerzetten, om zo de benodigde wendbaarheid te kunnen ondersteunen. Met kwaliteit wordt specifiek gedoeld op software die gebruikers in hun werkzaamheden ondersteunt, die eenvoudig en snel aanpasbaar is, en waarbij de functionele wensen herleidbaar zijn tot specifieke onderdelen in de software en andersom.

Zo'n gestructureerde aanpak wordt ook wel 'methode' genoemd, waarvan de eerste al uit de jaren '70 stammen. Vrijwel alle methoden voor softwareontwikkeling gebruiken modellen als tussenproduct, en zijn dan ook wel bekend als (methoden voor) modelgebaseerde (software) ontwikkeling. Door o.a. ontwikkelingen in technologie en de opkomst van nieuwe modelleertalen, zijn recente varianten van modelgebaseerde ontwikkeling ontstaan, waaronder model-gedreven architectuur, model-als-code en low-code en no-code platforms. Hoewel sommige van deze ontwikkelingen recent zijn, lijkt er (nog) geen methode te zijn die aan de

[6] Onder organisaties worden zowel commerciële bedrijven als (semi-)publieke instanties en samenwerkingsketens verstaan.

geïdentificeerde kwaliteitseisen voldoet om de wendbaarheid van organisaties te ondersteunen of zelfs te verbeteren. Het doel van dit onderzoek is om een basis te leggen voor een dergelijke methode, waarin vanuit organisatiemodellen software wordt ontwikkeld. Om dat doel te bereiken is een onderzoeksuitdaging als doel geformuleerd met zeven onderzoeksvragen.

Aanpak

De droom achter dit onderzoek is om een methode te hebben, ondersteund door de benodigde (IT) hulpmiddelen, om vanuit modellen van de organisatie software te ontwikkelen die voldoet aan de genoemde kwaliteitseisen, meermaals kan worden toegepast en aangepast kan worden aan specifieke situaties. Om zo'n methode en hulpmiddelen te ontwikkelen, zijn precieze specificaties nodig. Tegelijkertijd zijn er nog onzekerheden m.b.t. het ontwikkelen van een dergelijke methode, waaronder welke modellen het beste gebruikt kunnen worden en of zo'n aanpak überhaupt mogelijk is.

Vanwege deze onzekerheden is dit onderzoek gepositioneerd als een verkennend onderzoek, een noodzakelijke fase waarin een probleemgebied wordt verkend, parallel met het opbouwen van een grove oplossingsrichting. Ook worden in deze fase aannames getoetst en de specificaties helder gemaakt. Dit onderzoek richt zich primair op het wegnemen van de onzekerheden, terwijl een eerste oplossingsrichting wordt geschetst en getoetst.

Methoden zijn vaak het resultaat van het structureren of abstraheren van in de praktijk (formeel of informeel) gehanteerde procedures of aanpakken. Om de onzekerheden weg te nemen in deze verkennende fase, en de onderzoeksvragen te beantwoorden, moeten onderzoek in de praktijk worden gedaan. Action Design Research (ADR) is de combinatie van Design Science Research en Action Research, gericht op het degelijk en gestructureerd aanpakken van praktisch onderzoek. ADR definieert vier fasen die gericht zijn op zowel het oplossen van een (potentieel) probleem in de praktijk alsook het creeëren van een wetenschappelijke bijdrage.

De onderzoeksvragen worden initieel beantwoord met een literatuurstudie. Deze initiële antwoorden worden daarna getoetst in de praktijk met probleemgedreven onderzoek. Voor het praktijkonderzoek zijn vier verkennende studies gedaan, die allemaal gericht zijn op het ontwikkelen van een mogelijk onderdeel van de gehele methode. Door het combineren van praktisch en theoretisch onderzoek, in meerdere iteraties en met meerdere praktijkstudies, is de relevantie en (wetenschappelijke) onderbouwing van dit onderzoek aangescherpt.

Resultaten

In dit onderzoek is een model-gedreven aanpak tot softwareontwikkeling geadopteerd, waarin code wordt gegenereerd vanuit organisatiemodellen. Voordelen van deze aanpak zijn primair de snelheid en de traceerbaarheid. Voor de organisatiemodellen is gekozen voor Design and Engineering Methodology for Organizations

(DEMO), dat de essentie of ontologie van een organisatie blootlegt. Omdat een DEMO model onvoldoende details bevat voor het volledig genereren van software, is het Organisatie Implementatie Raamwerk (OIR) ontwikkeld dat voorschrijft om inrichtingskeuzes als waarden voor zekere Organisatie Implementatie Variabelen (OIVs) vast te leggen. Het ontwikkelde raamwerk maakt het mogelijk bewust de dimensies te kiezen waarin een organisatie wendbaar dient te zijn, terwijl andere dimensies als minder belangrijk kunnen worden aangemerkt – beiden hebben hun eigen weerslag in de software. Door gebruik te maken van DEMO en het OIR is in grote mate gewaarborgd dat alle en niet meer dan de relevante klantvereisten als startpunt worden genomen, aangevuld met, waar nodig, specifieke vereisten op het gebied van wendbaarheid. Een risico van deze aanpak is dat complexiteit wordt verplaatst van de softwareontwikkeling naar het creëren van de organisatiemodellen. Omdat softwarecomplexiteit vaak het resultaat is van organisatiecomplexiteit, is het niet meer dan een logische stap om de complexiteit te brengen naar waar die (initieel) vandaan komt.

Het ontwikkelen van een (situationele) methode is afhankelijk van de beschikbaarheid van zogenaamde methode-onderdelen. Onderdelen van een methode kunnen worden geclassificeerd op drie assen: perspectief, abstract en granulariteit. Gerelateerd onderzoek is bestudeerd om bestaande onderdelen voor het ontwikkelen van software vanuit een DEMO model te classificeren. Hoewel er meerdere onderdelen zijn geïdentificeerd, blijkt helaas geen van de bestaande onderdelen een volledig DEMO te ondersteunen, slaan vrijwel alle onderdelen de stap naar implementatie geheel over, en is er slechts één bestaand technisch onderdeel gevonden voor het genereren van software vanuit DEMO.

Voor het uitvoeren van de verkennende studies, zijn vier verschillende en moderne technologieën geselecteerd, gedreven door de klantorganisaties die een concrete uitdaging hadden binnen de scope van dit onderzoek. De geselecteerde technologieën, die tevens (een significant deel van) de kwaliteitseisen afdekken, zijn microservices, mockups (wireframes), Normalized Systems en low-code. Het gemeenschappelijke deel van deze verkennende studies is dat ieder start vanuit een DEMO model, in meer of mindere mate aangevuld met inrichtingskeuzes. In alle gevallen is vanuit deze modellen middels een algoritme of vertaaltabel software ontwikkeld. In sommige gevallen zijn deze algoritmes of vertaaltabellen volledig geautomatiseerd, waardoor geautomatiseerde software-generatie mogelijk is gemaakt. De ontwikkelde algoritmes, potentiële onderdelen voor een volledige methode, zijn typisch getoetst op meerdere organisaties, om de relevantie alsook (her)bruikbaarheid van de onderdelen te borgen. De verkennende studies hebben aangetoond dat het mogelijk is om op gestructureerde en, dus, automatiseerbare wijze vanuit een DEMO model, mogelijk aangevuld met inrichtingskeuzes, software te ontwikkelen.

Samengevat vormen de geïdentificeerde en gecreëerde onderdelen de basis om de gehele methode te gaan ontwikkelen. De specificaties van een dergelijke methode, die moet voldoen aan diverse kwaliteitseisen, zijn gevalideerd en aangevuld. Vanuit de specificaties kan met de onderdelen een (specifieke of generieke) methode worden samengesteld om – uiteindelijk – de wendbaarheid van een organisatie te kunnen ondersteunen. Het onderzoek heeft laten zien dat er vraag

is naar een dergelijke methode die uiteindelijk de kosten van softwareontwikkeling kan reduceren, terwijl deze minder afhankelijk is van (technisch geschoolde) softwareontwikkelaars.

Beperkingen en toekomstig onderzoek

De verkennende studies zijn uitgevoerd voornamelijk bij (semi)publieke organisaties in Nederland. Hoewel er geen reden is om aan te nemen dat de resultaten niet ook van toepassing zijn bij andere organisaties en buiten Nederland, kan alleen breder onderzoek deze aanname valideren.

De algoritmes die zijn ontwikkeld in de verkennende studies zijn veelal gedeeltelijk handmatig uitgevoerd. Een van de redenen is eenvoudigweg het gebrek aan een goed digitale hulpmiddel voor het maken van DEMO modellen dat het model tevens kan exporteren naar een door software leesbaar formaat. Om de algoritmes verder te automatiseren en het risico op het introduceren van fouten door handmatige stappen te reduceren, is het noodzakelijk dat dergelijke hulpmiddelen worden (door)ontwikkeld.

Voor een bredere adoptie van de ontworpen methode is het wenselijk deze onderdeel te maken van bestaande methoden. Specifiek lijkt het integreren van taalmodellen[7] en Kunstmatige Intelligentie voor het sneller en beter opzetten van DEMO modellen een interessante stap. Om methoden te integreren is het belangrijk dat ze eenzelfde theoretische basis hebben. Nader onderzoek is nodig om aanpakken vergelijkbaar te maken.

Hoewel IT steeds belangrijker wordt voor de operatie van organisaties, is wendbare software slechts één onderdeel in het bereiken van een wendbare organisatie. Gerelateerd onderzoek heeft aangetoond dat het mogelijk is om vanuit architectuurprincipes de wendbaarheid van organisaties te ondersteunen. Tegelijkertijd is en blijft de vraag voorlopig welke constructies er zijn, al dan niet op concepten uit DEMO gebaseerd, die voldoen aan al die principes. Om dergelijke constructies te kunnen vergelijken, dient wendbaarheid van organisaties meetbaar te zijn. Hoewel een basis is gelegd voor de ontwikkeling van wendbare software ter ondersteuning van wendbare organisaties, is er nog veel aanvullend onderzoek nodig om organisaties wendbaar te maken in alle relevante aspecten.

[7] Bijvoorbeeld ChatGPT.

Contents

Publications

The publications below have contributed to the contents of this thesis. A detailed mapping from the most contributing papers to this thesis is included.

Conference and Journal Publications

EEWC 2011 M. R. Krouwel and M. Op 't Land, *Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems*, in EEWC 2011: Advances in Enterprise Engineering V, A. Albani, J. L. G. Dietz, and J. Verelst, eds., vol. 79 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2011, pp. 31–45. 10.1007/978-3-642-21058-7_3

PRET 2011 M. Op 't Land, M. R. Krouwel, E. van Dipten, and J. Verelst, *Exploring Normalized Systems Potential for Dutch MoD's Agility – A Proof of Concept on Flexibility, Time-to-Market, Productivity and Quality*, in PRET 2011: Practice-Driven Research on Enterprise Transformation, F. Harmsen, K. Grahlmann, and E. Proper, eds., vol. 89 of Lecture Notes in Business Information Processing, Springer, Sept. 2011, pp. 110–121. 10.1007/978-3-642-23388-3_5

MCIS 2012 M. R. Krouwel and M. Op 't Land, *Using Enterprise Ontology as a basis for Requirements for Cross-Organizationally Usable Applications*, in Proceedings of the 7^{th} Mediterranean Conference on Information Systems 2012 (MCIS2012), A. D. Figueiredo, I. Ramos, and E. Trauth, eds., MCIS Proceedings, University of Minho, Portugal, Sept. 2012, AIS Electronic Library (AISeL). Paper 23. http://aisel.aisnet.org/mcis2012/23

EEWC 2013 M. Op 't Land and M. R. Krouwel, *Exploring Organizational Implementation Fundamentals*, in EEWC 2013: Advances in Enterprise Engineering VII, H. A. Proper, D. Aveiro, and K. Gaaloul, eds., vol. 146 of Lecture Notes in Business Information Processing, Springer-Verlag Berlin Heidelberg, 2013, pp. 28–42. 10.1007/978-3-642-38117-1_3

EEWC 2016 M. R. Krouwel, M. Op 't Land, and T. Offerman, *Formalizing Organization Implementation*, in EEWC 2016: Advances in Enterprise Engineering X, D. Aveiro, R. Pergl, and D. Gouveia, eds., vol. 252 of Lecture Notes in Business Information Processing, Funchal, Madeira Island, Portugal, 2016, Springer, pp. 3–18. 10.1007/978-3-319-39567-8_1

EEWC 2016 L. de Laat, M. Op 't Land, and M. R. Krouwel, *Supporting Goal-oriented Organizational Implementation - Combining DEMO and Process Simulation in a Practice-tested Method*, in EEWC 2016: Advances in Enterprise Engineering X, D. Aveiro, R. Pergl, and D. Gouveia, eds., vol. 252 of Lecture Notes in Business Information Processing, Springer, 2016, pp. 19–33. 10.1007/978-3-319-39567-8_2

EEWC 2020 M. Op 't Land, M. R. Krouwel, and S. Gort, *Testing the Concept of the RUN-Time Adaptive Enterprise - Combining Organization and IT Agnostic Enterprise Models with Organization Implementation Variables and Low Code Technology*, in EEWC 2020: Advances in Enterprise Engineering XIV, D. Aveiro, G. Guizzardi, R. Pergl, and H. A. Proper, eds., vol. 411 of Lecture Notes in Business Information Processing, Springer, Apr. 2021, pp. 228–242. 10.1007/978-3-030-74196-9_13

EEWC 2021 M. R. Krouwel and M. Op 't Land, *Business Driven Micro Service Design - An Enterprise Ontology based approach to API Specifications*, in Advances in Enterprise Engineering XV, D. Aveiro, H. A. Proper, S. Guerreiro, and M. De Vries, eds., vol. 441 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 95–113. 10.1007/978-3-031-11520-2_7

PoEM 2022 M. R. Krouwel, M. Op 't Land, and H. A. Proper, *Generating Low-Code Applications from Enterprise Ontology*, in PoEM 2022: The Practice of Enterprise Modeling, B. S. Barn and K. Sandkuhl, eds., vol. 456 of Lecture Notes in Business Information Processing, Springer Nature Switzerland AG, 2022, pp. 19–32. 10.1007/978-3-031-21488-2_2

SoSyM M. R. Krouwel, M. Op 't Land, and H. A. Proper, *From Enterprise Models to Low-Code Applications: Mapping DEMO to Mendix, illustrated in the Social Housing domain*, International Journal on Software and Systems Modeling, (2024). Invited submission, currently under review

Other Publications

M. R. Krouwel, *Understanding the Essence of Organizations with DEMO-4*. Online, Jan. 2021. https://www.linkedin.com/pulse/understanding-essence-organizations-demo-4-marien-krouwel/

M. R. Krouwel, M. Stam, A. Bulat, and R. de Wit, *Mendix and SAP: do you choose or combine?* Online, Aug. 2021. https://www.linkedin.com/pulse/mendix-sap-do-you-choose-combine-marien-krouwel/

M. R. Krouwel, *Low Code and No Code: Is There a Difference? A practical analysis*. Online, Feb. 2022. https://www.linkedin.com/pulse/low-code-difference-practical-analysis-marien-krouwel/

J. Dietz, E. van Dipten, M. Krouwel, P. Kuipers, T. de Mik, and I. Theuwissen-Krol, *Begrijpen en Maken van Essentiële Modellen*, Werken met DEMO (WmD), Enterprise Engineering Institute, Nov. 2021. Dutch. https://ee-institute.org/demo/werken-met-demo/

J. L. G. Dietz, M. Op 't Land, M. R. Krouwel, and J. B. F. Mulder, *Enterprise Design*, Springer, 2024. Forthcoming

Mapping From Papers to Thesis

Parts of this thesis have been published as separate papers. This section provides an overview of how the original papers were used in this thesis.

M. R. Krouwel and M. Op 't Land, *Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems*, in EEWC 2011: Advances in Enterprise Engineering V, A. Albani, J. L. G. Dietz, and J. Verelst, eds., vol. 79 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2011, pp. 31–45. 10.1007/978-3-642-21058-7_3

Section in paper	Section in thesis	Changes
1. Introduction	Section 8.1	
2. Theoretical Background	Section 4.2, Section 3.4.3	Updated to DEMO-4 and 2016-version of NS theory
3. Theoretical Comparison	Section 8.2	
4. Enterprise Ontology of the Dutch Governmental Subsidy Schemes	Section 8.2	Updated to DEMO-4
5. Deriving a Normalized System from an Enterprise Ontology	Section 8.2	Updated to DEMO-4
6. Implementation Freedom	Section 8.2	
7. Conclusions and Future Research	Section 8.3	
n/a	Section 8.4	Added identified method fragments

M. R. Krouwel and M. Op 't Land, *Using Enterprise Ontology as a basis for Requirements for Cross-Organizationally Usable Applications*, in Proceedings of the 7^{th} Mediterranean Conference on Information Systems 2012 (MCIS2012), A. D. Figueiredo, I. Ramos, and E. Trauth, eds., MCIS Proceedings, University of Minho, Portugal, Sept. 2012, AIS Electronic Library (AISeL). Paper 23. http://aisel.aisnet.org/mcis2012/23

Section in paper	Section in thesis	Changes
1. Introduction	Section 7.1	
2. Research Design	Section 3.2, Section 4.2, Section 7.2	Updated to DEMO-4
3. The Approach in Practice	Section 7.2	
4. Results (Case Level)	Section 7.2	Updated to DEMO-4
5. Conclusions and Discussion (Research Level)	Section 7.3, Chapter 11	
n/a	Section 7.4	Added identified method fragments

M. R. Krouwel, M. Op 't Land, and T. Offerman, *Formalizing Organization Implementation*, in EEWC 2016: Advances in Enterprise Engineering X, D. Aveiro, R. Pergl, and D. Gouveia, eds., vol. 252 of Lecture Notes in Business Information Processing, Funchal, Madeira Island, Portugal, 2016, Springer, pp. 3–18. 10.1007/978-3-319-39567-8_1

Section in paper	Section in thesis
1. Introduction	Section 1.1, Section 4.2
2. Way of Thinking	Section 3.2, Section 4.2
3. Way of Working	Chapter 2
4. Result	Section 4.3
5. Conclusions and Future Research	Section 4.3, Chapter 11

M. R. Krouwel and M. Op 't Land, *Business Driven Micro Service Design - An Enterprise Ontology based approach to API Specifications*, in Advances in Enterprise Engineering XV, D. Aveiro, H. A. Proper, S. Guerreiro, and M. De Vries, eds., vol. 441 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 95–113. 10.1007/978-3-031-11520-2_7

Section in paper	Section in thesis	Changes
1. Introduction	Section 3.4.2, Section 6.1, Section 6.2	
2. Research Design	Chapter 2	
3. Foundations on Enterprise Ontology and Service Design	Section 4.2, Section 5.2, Section 6.2	
4. Devising the Algorithm	Section 6.2	Added some details that did not fit in the original paper
5. Evaluation of the Algorithm	Section 6.2, Section 6.3	Minor changes
6. Conclusions and Future Research	Section 5.2, Section 6.2, Section 6.3	Minor changes
n/a	Section 6.4	Added identified method fragments

M. R. Krouwel, M. Op 't Land, and H. A. Proper, *Generating Low-Code Applications from Enterprise Ontology*, in PoEM 2022: The Practice of Enterprise Modeling, B. S. Barn and K. Sandkuhl, eds., vol. 456 of Lecture Notes in Business Information Processing, Springer Nature Switzerland AG, 2022, pp. 19–32. 10.1007/978-3-031-21488-2_2

Section in paper	Section in thesis	Changes
1. Introduction	Section 1.1, Section 3.3, Section 3.4.4, Section 4.1.3, Section 5.2, Section 9.1	
2. Theoretical Background	Section 3.3.2, Section 3.4.4, Section 4.1.3, Section 4.2, Section 4.3	
3. Mapping	Section 9.2	Updated to latest version of both metamodels
4. Implementation and Evaluation	Section 9.2	Added and updated some screenshots
5. Conclusions and Discussion	Section 9.3, Section 9.4, Chapter 11	
n/a	Section 9.4	Added identified method fragments

The authors of this paper were invited to submit an extended version to SoSyM. This extended version [267], including more detailed parts of this thesis, has been submitted and is currently in the review process (final round).

"(...) the species that survives is the one that is able best to adapt and adjust to the changing environment in which it finds itself"

Leon C. Megginson [310, p. 4]

"Blessed are the flexible, for they will not be bent out of shape."

Robert Ludlum, in 'The Janson Directive' (2002)

"Program testing can be used to show the presence of bugs, but never to show their absence."

Edsger W. Dijkstra [127]

1 Introduction

This thesis is about creating evolvable software for (agile) enterprises[1] in a structured way. The quotes above may seem a bit off-topic, but are close to my heart and, combined, they form the core of the motivation of this thesis.[2]

In 1963, the first quote[3] was said to be applicable not only to individuals but also to institutions, businesses and even to civilizations. Therefore, enterprises, including their supporting software, need to be agile in order to survive.

The second quote, originally from an espionage thriller in the context of changing a plan when needed, is currently used a lot in yoga practice to stress the importance of a flexible mind and body. It shows, again, that flexibility and agility is needed in multiple aspects, for both individuals and enterprises.

The last quote criticizes most current approaches to software development that involve multiple manual conversions from requirements to code, risking the introduction of defects in each conversion step. These approaches rely heavily on software testing to improve or maintain its quality, but it is never possible to show that no defects were introduced during development. As numbers show that solving a defect (or: bug) in production can be up to 100 times more expensive than solving it in the design phase [95], it is highly undesired that defects are introduced late(r) in the software development life cycle. Applying a more mathematical approach to software development, e.g., by applying model transformation(s), has the advantage that it is possible to *a priori* show by means of formal reasoning that the applied transformation does not introduce (new) defects. Not introduc-

[1] An enterprise is defined as any goal oriented cooperative, including commercial businesses, public organizations and non-profit institutions as well as chains of such organizations.

[2] For the connection between the quotes used in this thesis and my personal life, the reader is referred to my resume on p. 273.

[3] Although this quote originates from Charles Darwin's evolutionary theory, it is not from Darwin and often incorrectly attributed to him.

ing defects during software development is even more important when you need to change software often.

The sum of these quotes brought the dream about having a structured way to quickly and traceably convert *enterprise models*, describing its (essential) user activities and *known (*or *desired) flexibility*, to working and *adaptable software*, without defects, so that software (development) is not the limiting factor in changing an enterprise. Using model transformations – that can be expressed as mathematical functions or algorithms – in the process of software development not only could be an answer to this dream, but also has the potential to close the gap between software design, development and maintenance, and therefore could reduce the (big) difference in effort and related costs needed to fix a bug in production compared to fixing it during the design phase.

The remainder of this chapter is structured as follows: in Section 1.1 the concept of enterprise agility is introduced, needs in software development to support enterprise agility – and, thus, enterprise survival – are formulated and partial answers to the formulated needs are outlined; in Section 1.2 the research gap is detailed which is then used to formulate the research challenge, questions and deliverables in Section 1.3. While the research approach to answer the research questions will follow in Chapter 2, the outline of this thesis is provided in Section 1.4.

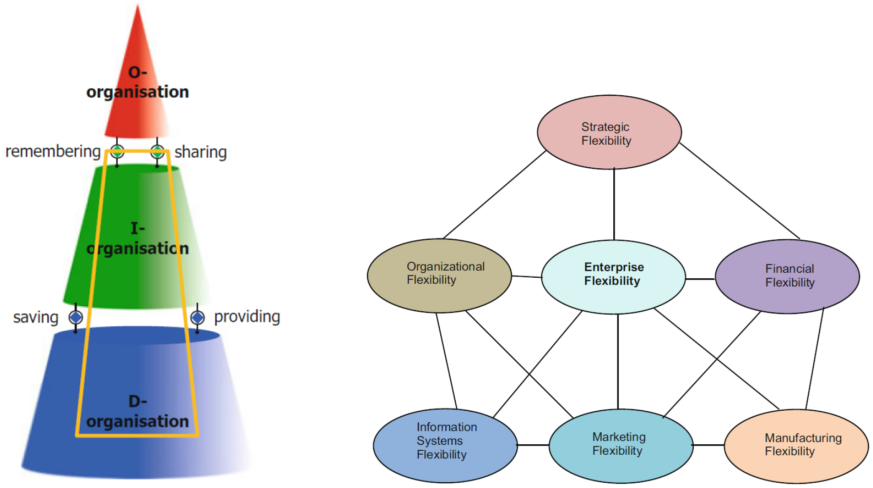
1.1 Background

Due to factors such as hyper-competition [93], increasing expectations from customers, regulatory changes, and technological advancements, the conditions in which enterprises need to thrive become increasingly turbulent. As a result, the ability to change with an ever decreasing time-to-market, often referred to as 'agility' [470], becomes an important determinant for the success of enterprises [451, 348]. In a society where most enterprises are supported by software, unfortunately, software is seen more as obstruction to than as enabler for enterprise agility [471]. Because an enterprise and its supporting software are in fact intrinsically intertwined [124, p. 251] (see Figure 1.1a), enterprise software has to be able to quickly respond and change directions to support enterprise agility [392].

In this section the notion of enterprise agility will be explored, and refined into specific needs in software development that are considered critical in achieving enterprise agility. The use of methods and models, together known as Model-based Engineering (MBE) (see Section 3.3), is shown to be a partial answer in order to identify the research gap.

1.1.1 Enterprise Agility

Agile enterprises are able to adapt rapidly and cost efficiently in response to environmental changes [455, 128]. It is often operationalized as a set of organizational capabilities, behaviors, and ways of working that affords their business



(a) Notion of Enterprise Information Systems [124]

(b) Cornerstones of enterprise flexibility [437]

Figure 1.1: As software is an inseparable part of an enterprise, software agility and enterprise agility are strongly connected

freedom, flexibility, and resilience to achieve its purpose [67]. This 'dynamic capability' [441] can be further disaggregated into two key capabilities [327, 399, 306, 470, 379, 196, 44]:

a) *sensing* the environment for (relevant) changes and detecting flaws in current operations, and

b) *responding* to these (external and internal) factors by deciding whether the internal organization needs to be changed at all and, when required, to implement the change.

The notion of 'agile enterprise' is also referred to as the 'run-time adaptive enterprise' [342] or the 'flexible enterprise' [437]. Different aspects of flexibility can be identified that can be considered cornerstones in creating a flexible enterprise, including strategic flexibility, organizational flexibility, financial flexibility, marketing flexibility, manufacturing flexibility and (automated) Information System (IS)[4] flexibility [437, Figure 1.4] (see Figure 1.1b). Nowadays, enterprises attempt to be agile in all of these aspects [42]. As a result, enterprises are continuously being redesigned [173] and enterprise engineering (including software development) is an ongoing activity that requires continuous alignment between the business (activities) and its supporting software [213].

[4]In this thesis, whenever there is a reference to Information System, it is meant in the sense of the automated part of an IS that uses Information Technology (IT) or, more specifically, Information and Communication Technology (ICT) and typically consists of hardware, software and network components.

Reflection on Current Approaches

In achieving enterprise agility, there seem to be two distinct but related axes: *product* and *process* [175, 257, 423, 296, 224]. When the product is (easily) adaptable, but the processes to sense and respond to changes are not in place, chances are little that the product is adapted to changing needs on time. On the other hand, if the processes are there, but the product is not easily adaptable, changing the product will still take a lot of time. For example, a closet with fixed shelves is harder to change, if possible at all, than a closet with a flexible shelf system in case you need to make space for a book that is taller than the current shelf height [45] (see Figure 1.2).

(a) Cupboard with fixed shelves

(b) Cupboard with flexible shelf system

Figure 1.2: Fixed versus flexible product

For agility in software development, SCRUM[5] and DevOps[6] are well-adopted (best) practices [302, 235, 78, 422, 64, 100, 172, 283]. At the same time there seem to be many issues with their adoption [6, 235]. It seems that these practices

[5] https://www.scrum.org/

[6] https://en.wikipedia.org/wiki/DevOps

are mainly focused on introducing the processes to support enterprise agility[7] and thus do not ensure that necessary product changes can be executed within limited amount of time – almost everyone is familiar with the time it can take to (re)configure standard packages or to create or change custom-made software to support changing business requirements.

In order to accommodate (software) product flexibility, best practices include modularization [12, 223, 362]. However, finding the right level of modularization as well as defining the exact scope of a module, turns out to be hard in practice [72, 30]. Applying a microservice architecture is currently a popular way to modularize an application landscape, but suffers from similar issues [492] (see Section 3.4.2).

Summarizing, (business) *processes* to make sure an enterprise is *aware* of changes that need to be dealt with is not enough; additionally there is a need for *structures* – in terms of organization and supporting software – that allow to *implement* changes quickly, supported by processes to ensure those structures are adhered to. In the next section specific needs regarding both the process to develop software and the structure of the resulting software will be formulated.

1.1.2 Needs in Software Development

The increased required flexibility as well as the ever-increasing complexity of both enterprises and supporting software make that IT budgets have to grow every year, as was stated by Lehman in 1980 [281]: "As an evolving (software) program is continuously changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it." It is no surprise that most enterprises have to increase their IT budgets every year [83, 170] as Lehman's law implies that enterprises that do not increase the yearly IT budget will be faced with less satisfactory IT and decreased support of organizational changes.

The driver of this research is the dream to have an approach towards software development that supports the quick and continuous creation and adaptation of high quality software solutions to support an enterprise's agility – and, of course, its employees and customers. Based on practical experience, some specific quality attributes are considered key to support this dream. First the broader topic of software quality is introduced, and then the specific needs are formulated: supporting the end users, evolvability, speed, and traceability.

Quality

Software quality can be measured on two main axes: *a) functional* quality is about how well it complies to its functional requirements [88], and *b) structural* quality is about how well it is structured and meets non-functional requirements [232]. Many frameworks and models exist to define and/or measure software (structural) quality. Cavano and McCall [75] mention correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, and interoperability. The Evans and Marciniak factor model [136] and the

[7] Some even say that SCRUM has nothing to do with agile at all, see, e.g., http://www.denniswyland.net/blog/?p=43.

Deutsch and Willis factor model [112] mention similar factors, but add verifiability, expandability, safety, manageability, and survivability [168]. Blundell et al. further extend this list and propose a list of software quality metrics [47]. Davis proposes a total of 30 principles for quality in software development [94]. Some of these dimensions are now part of ISO/IEC 25010.[8] It shows that quality is important but at the same time hard to achieve. It has been stated that up to 50% of all time spent in software development goes to testing and debugging [499]. As software quality is such a broad topic, a specific need cannot be formulated. Instead, some specific quality aspects that are considered relevant for this research are highlighted in the next paragraphs.

Supporting the End Users

In contrast to, e.g., neurological systems, biological systems, society as a whole and the world economy, that can be considered emergent systems that just change while they are being used without explicit design upfront, enterprises and their supporting software systems are considered deliberately designed systems.[9,10] The development of such deliberately designed systems, and software systems specifically, starts with the users needs, or requirements [120].[11] It is thus a quality aspect of the functional kind. There is an increasing demand for software that matches real user needs in a working environment [36]. It is no longer sufficient to (just) deliver products of technical excellence; instead, products need to be useful and usable for consumers and professional users [269]. The Agile Manifesto[12] emphasizes the collaboration with customers, but its approach does not guarantee that the software development team will work towards solving the correct problem [330]. In a sense, it is not (only) about building *it right*, but about building the *right it* [143]; if the software doesn't support its users in their daily tasks, it's either not useful, or not usable, or fails on both aspects.

The biggest issue in identifying requirements is that talking to users might not be enough to reveal their true needs [249, 231]. Or, even bolder: "People don't know what they want until you show it to them" (attributed to Steve Jobs). Although early involvement of users [271] and Design Thinking approaches, e.g., [61, 477, 290], can help in creating satisfying software solutions, they still rely for a big extent on asking users what they need. Instead, user activities should be described in such a way that all user requirements are there, and not more than that – i.e., they need to be complete and concise – without relying on users

[8] ISO/IEC 25010:2011 is the successor of ISO 9126 and can be found at https://www.iso.org/standard/35733.html.

[9] Etymologically, system means 'organized whole', from the Latin 'systema'.

[10] Emergent Software Systems are software systems that build their own understanding of their environment and autonomously optimize at run-time by composing discovered components [361, 147]. These systems usually involve forms of Artificial Intelligence (AI) and therefore still need to be designed at some level. Also, most development practices nowadays involve iterations where (end user) feedback is being used to design the next version of the system. In this research, applying iterative development is considered as emergence within deliberately designed systems.

[11] A detailed explanation can be found in Section 3.2.

[12] https://agilemanifesto.org/

solely. The first need is formulated as follows:

> **Need 1:** *The set of user requirements for the software system needs to be complete and concise.*

Preferably these requirements are captured and expressed in a way that is easily understandable and verifiable by end users.

Evolvability

Software evolvability, the ability to easily and cost effectively accommodate changes, is considered an important software quality aspect [80]. In the context of enterprise agility, software needs to be adaptable or flexible in order to support changing business requirements [143]. In 1969, McIlroy envisioned that software components should be easily replaceable [308], which is why the next need is formulated as follows:

> **Need 2:** *The software system should be easily adaptable.*

A potential issue in achieving software adaptability, is Lehman's law that states that the amount of work required to replace or change components, will grow over time [281]. In a response, Normalized Systems theory has formulated a set of anticipated changes on the software level and suggest using the concept of system stability as starting point for software design [299].[13]

As the scope of this research is not just any software, but *enterprise* software specifically, it is considered important to look at anticipated changes at the *enterprise* level, that could or should be supported by software, and find stability at the enterprise level before looking at how to support that with software. An enterprise can be described at three different levels of abstraction: what it does, how it does it, and with what it does that [501, 474]. Based on these three abstraction levels, the following kinds of changes relevant for enterprises are considered:

- adding or deleting[14] a product or service;
- adding or deleting a department, functionary type, means type or process step; and
- adding or deleting a specific technological resource, such as a human-being, a piece of software or a specific mean.

Speed

As said in the introduction, the world is ever-changing. This implies that software needs to be created and adapted at a speed that is not lower than the desired or required rate of change at business level. Keeping up with competition by reducing the time-to-market of new or changed products is still considered a

[13] More about software evolvability and Normalized Systems theory can be found in Section 3.4.3.

[14] A change is seen as the combination of a delete and add action.

dominant issue [499]. Although CI/CD pipelines, reusing components, microservice architectures, different ways of working and new management styles support the release of new pieces of software within minutes [46, 27], it is only a matter of time before software development is slowed down again [282]. In fact, this calls for a (fundamentally) different approach towards software development. The need is formulated as follows:

> **Need 3:** *It should be able to create (a new version of) the software system in little time.*

Traceability

Traceability refers to the degree to which a relationship can be established between two or more products of the development process [221]. It implies that requirements are clearly linked to design artifacts, source code, test cases, etc. [369], preferably bidirectional [276], and therefore tries to prevent hidden design decisions in the creation of software, something that, unfortunately, still happens quite often [192, 494, 458]. Traceability allows *a)* to validate whether all requirements have been implemented [383, 326], *b)* to provide information on the justification, decisions and assumptions behind a requirement [369], *c)* for easy determination of the impact of a change [396, 383], and *d)* to understand why a software artifact was created [254]. It has been recognized as an important factor in the quality of a software development product [360]. Although much research has been done in the area of traceability in software development, there are still many open issues [426, 326]. This leads to the last need.

> **Need 4:** *It should be clear, i.e., traceable, how a requirement is implemented in the software system.*

1.1.3 Partial Answers

Applying a structured method in creating software has shown to improve efficiency [151] and quality [245] and hence improves the discipline of software engineers [192]. A method towards software development[15] is a step-by-step approach, based on a specific way of thinking, consisting of structured activities and deliverables and is preferably supported by tools [17, 58, 199]. More specifically, a method should contain a Way of Thinking (WoT), a Way of Working (WoW), a Way of Modeling (WoM), Way of Controlling (WoC), and a Way of Supporting (WoS) [411, 488, 487, 176, 87, 92] (see Figure 1.3).

Already in the 1970s, just shortly after Dijkstra's discussion on step-wise computer program composition [127], the first structured methods for analyzing, designing and developing software systems have emerged, e.g., Structured Analysis and Design Technique (SADT) [384, 500, 109] and Structured System Design (SSD) [154]. These methods all rely on the use of models as (intermediate) deliverables, such as Entity-Relationship Diagrams (ERDs) [79], flowcharts [500],

[15] Also known as Information Systems Development Method (ISDM). In this thesis the term *method* is used to refer to an ISDM.

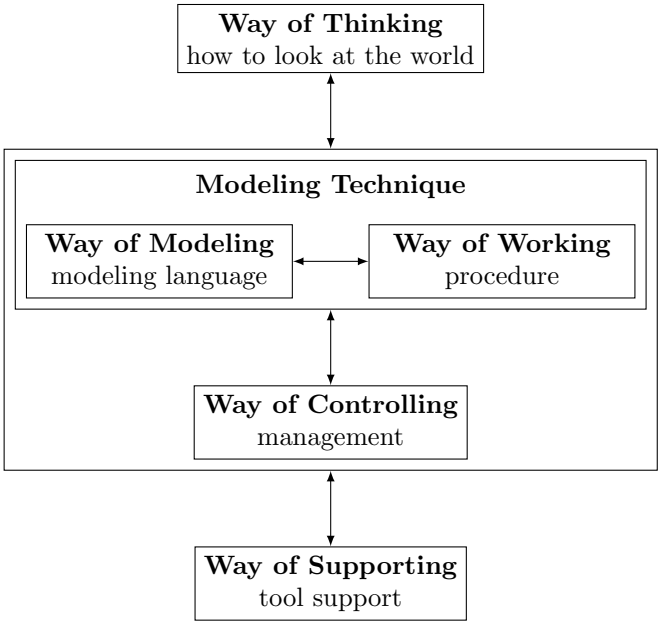


Figure 1.3: Aspects of a method, adapted from [411, 488, 487, 176, 237, 92]

Data Flow Diagrams (DFDs) [109], State Transition Diagrams (STDs) [498] and IDEF [307]. Combining different modeling techniques (see Figure 1.3) is key to improving the software development process [24].

A model is a simplified representation of a (software) system, that is used to study aspects of the system one is interested in [357, 291, 408, 142, 270, 124] and are key to support human understanding, reasoning and communication [394]. Models allow representing and studying systems even prior to implementation [479, 410] and are being used to explain and validate the intended behavior of a (software) system [410, 203]. Therefore, models are an important input for the design and development of software [479, 295, 76]. Not using models is considered inadequate for creating and maintaining large-scale or complex software systems [60].

Using (formal) models in the process of system development is also known as Model-based Engineering (MBE), and the Computer-aided Software Engineering (CASE) tools from the 1980s can be considered as a first implementation of MBE [160, 292, 285]. These tools quickly evolved into tools to support Object-Oriented Programming for the web, aimed at integrating with other (online) systems. A more recent and well-known implementation of MBE is Model-driven Architecture (MDA),[16] a software development approach that relies heavily on the use of Unified Modeling Language (UML) models [407]. However, MDA comes with a lot of issues, including poor integration of different models, lack of efficiency, limited applicability, vendor lock-in, and its complexity [419, 331].

[16] https://www.omg.org/mda/

A reason for the perceived complexity of MBE might be that, even with the use of models, the wide conceptual gap between the enterprise itself and the supporting software system(s) is a big challenge in creating software for the enterprise and thus requires extensive manual work making software development complex and costly [160]. Others even say that models don't deliver *enough* value if they are not on a direct path to working software [430]. Despite its possible challenges, this research expects value in starting from high-level enterprise models that can be step-wise refined and translated into working software.

A reason that MBE now has more interest [403, 69] than CASE in the 1980s might be the advancements in the availability of technologies in general and (standardized) (enterprise) modeling techniques [192], as well as the availability of more advanced graphical interfaces do to the modeling [285]. The advancements on these aspects as well as marketing also seem the reason low code nowadays is an even bigger trend than MBE [69], while they can be considered synonyms [69, 50]. Currently there does not seem to exist an MBE approach towards software development that addresses all formulated needs, while the aforementioned advancements provide new possibilities to explore such a method.

1.2 The Need for a (New) Method

As outlined in the previous section, in order to support the agile enterprise, there is a need to be able to quickly create software of high quality that supports (traceably) real end users' needs and is adaptable to changing needs. Since an enterprise and its supporting software are closely related, the design of an enterprise and the design of its supporting software should not be separated. Since models cannot be left out of the software development process anymore, at least for creating (complex) enterprise software, the approach needs to start from high-level enterprise models that are step-wise refined and converted into working software. As the formulated needs are not unique to a specific enterprise, the aim is to have an approach that can be used repeatedly in different situations. Such an *explicitly structured* approach towards software development that addresses the formulated needs, is a *method*, as shown before.

Structuring an approach so that it can be applied repeatedly to different situations is called method engineering: the discipline to design, construct and adapt methods, techniques and tools for the development of software systems [272, 58]. Indeed, methods often result from structuring or generalizing procedures or approaches that are being applied in practice [199, 206] (see Figure 1.4).[17] In order for a method to be widely applicable, it should be adaptable to different situations [200]; this is known as a Situational Method, typically composed of several method elements.[18]

So, the dream that drives this research, is to have a method, including supporting tools, to create software from enterprise models that addresses the needs, can be applied repeatedly, and is adaptable to specific situations. In order to create

[17] The inverse is however also true [199, 3].

[18] More on (Situational) Method Engineering can be found in Section 5.1.

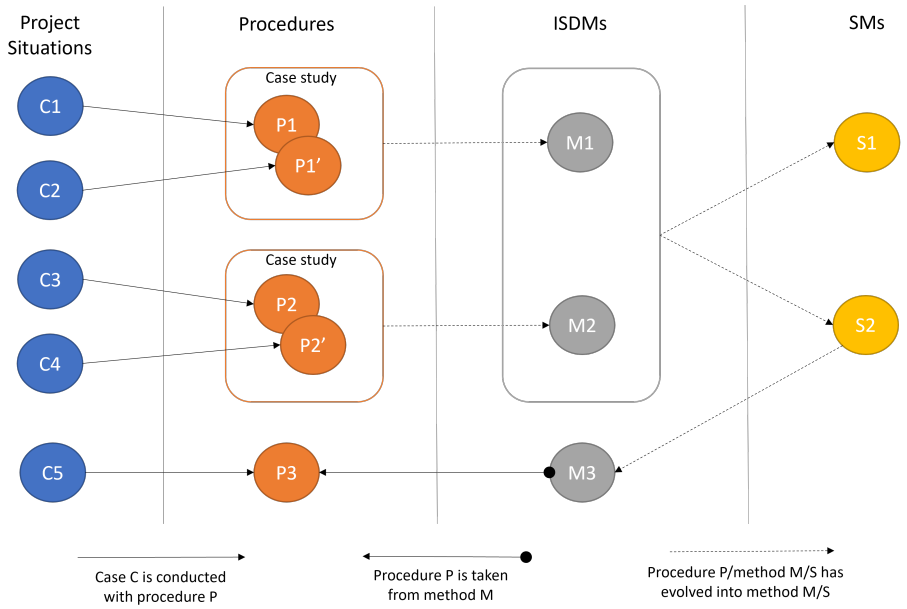


Figure 1.4: The creation of (situational) methods by structuring procedures that are applied on cases, adapted from [200]

such a method and supporting tools, clear specifications are necessary [367]. Although the needs defined in Section 1.1 can be considered an initial specification, there are some uncertainties:

- What are the requirements for such a method?
- What input model(s) should be chosen to support the step-wise refinement towards software?
- What target technology or technologies should be chosen?
- Is creating such a method technically feasible?
- How can the needs as defined in Section 1.1.2, i.e., completeness of requirements, flexibility in the solution, speed in the development process, and traceability from requirements to solution – i.e., no hidden design decisions – be supported?
- How adaptable should such a method be to meet different situations?

Due to these uncertainties, this research is considered part of the 'fuzzy front-end' [247] of creating a complete method. This fuzzy front-end is a necessary stage [253] in which a problem is explored, guided by a vague idea of the solution, and assumptions are being examined, resulting in a minimum specification [457] as well as an initial version of elements that could be part of the final solution [134]. This research will focus on reducing these uncertainties, while aiming to find *some* initial method elements that could be part of such a method.

1.3 Research Challenge, Questions and Deliverables

In this section the scope of this research in terms of a main research challenge will be detailed, supported by research questions and deliverables.

1.3.1 Research Challenge

In order to deal with the uncertainties for designing an integrated method for enterprise model-based software development, as outlined in Section 1.2, supporting the needs as outlined in Section 1.1.2, this research is considered an exploratory research that details how to create such a method. The main research challenge is as follows:

> **Main Research Challenge:** *How to create a method for the development of enterprise software that answers the identified needs?*

In order to deal with the main research challenge, several research questions are formulated.

1.3.2 Research Questions

As creating a method consists of multiple phases that come with many uncertainties, several research questions have been formulated to address these uncertainties. The research questions are outlined below, of which some are more theoretical while others are more practical. Addressing the identified needs is an implicit part of all questions.

Theoretical Research Questions

In order to choose a more specific approach for the structured and repeatable creation of software from enterprise models, the following research question is formulated:

> **Research Question 1** (RQ 1)**:** *What does it mean to step-wise create software for enterprises?*

In order to use enterprise models in the method, one or more model kind(s) need(s) to be selected as a starting point. The following research question guides the selection of the input model(s) or modeling technique(s) to provide the input model(s):

> **Research Question 2** (RQ 2)**:** *What model(s) should be used as input for the method?*

In order to create working software, one or more target technologies should be selected.

> **Research Question 3** (RQ 3)**:** *What target technology or technologies should be supported by the method?*

In order to be able to create a method after this fuzzy front-end research has been performed, it should detailed what creating a method means as well as how adaptable it needs to be.

> **Research Question 4** (RQ 4)**:** *What does the creation of a (situational) method entail?*

Practical Research Questions

The answers to the theoretical research questions should be validated in practice. Specifically, it should be validated if it is technically feasible to create software in the chosen target technology or technologies (RQ 3) from the chosen input model(s) (RQ 2).

> **Research Question 5** (RQ 5)**:** *Is it technically feasible to create software from (the selected) enterprise models?*

While the needs in Section 1.1.2 can be considered an initial set of requirements, only in practice it can be validated whether these are enough or whether there are others.

> **Research Question 6** (RQ 6)**:** *What are requirements for a method to develop software from enterprise models?*

One of the goals of the fuzzy front-end research phase is to create a first set of possible method elements for the method.

> **Research Question 7** (RQ 7)**:** *Which elements could be part of such a method?*

1.3.3 Deliverables

The theoretical contribution of this research is the founded selection of both input(s) and output(s) of the approach, in terms of enterprise models and target technologies. The practical contribution of this research is expected to be a set of possible method elements for creating software from enterprise models. In creating possible method elements a better view on the feasibility and requirements of such a method will be gained. Combined, these deliverables provide an overview of possible elements from both literature and practice, and gives guidance to later (possible) assembly a situational method. Moreover, this could be one of the first attempts to apply the theory of (Situational) Method Engineering in practice.

1.4 Outline

As explained in Chapter 2, a practice-driven ADR approach will be adopted to provide the answers to the research questions. ADR defines four stages: 1. problem formulation, 2. building, intervention and evaluation, 3. reflection and learning, and 4. formalization of learnings. This introduction has laid the foundation for the problem formulation on the research level. The following parts are structured according to the ADR stages (see Table 1.1):

- In order to further refine the research challenge and elaborate on the theoretical research questions, in Part I the research approach is detailed, the process of step-wise system and model-based software development is explored, the enterprise model(s) and the target technologies to work with are selected, and the concept of method engineering is explored in order to classify existing method elements. It is structured as follows:
  - In Chapter 2 the chosen research approach is detailed.
  - As both enterprises and software are systems to be designed, in Chapter 3 system development in general and software development specifically are explored, as well as the kind of models that can be used in the process, in an (initial) answer to RQ 1. Additionally, several modern software technologies are outlined in an (initial) answer to RQ 3.
  - In Chapter 4 the different kind of modeling techniques for enterprises. The choice for a specific modeling technique is made and more details on the chosen technique are provided in an (initial) answer to RQ 2.
  - The topic of (Situational) Method Engineering in Chapter 5 provides an (initial) answer to RQ 4. The provided framework is then used to classify existing method elements.
- In an attempt to create new possible method elements, that need to be built, evaluated and reflected on, and to validate the answers from Part I, Part II

Part/chapter	**ADR stage**	**Provides (initial) answers to**
Chapter 1	1. problem formulation	RQ 6
Part I	2. building, intervention and evaluation	
Chapter 2		
Chapter 3		RQ 1 and RQ 3
Chapter 4		RQ 2
Chapter 5		RQ 4
Part II	2. building, intervention and evaluation; and 3. reflection and learning	RQ 5, RQ 6, and RQ 7
Chapter 6		
Chapter 7		
Chapter 8		
Chapter 9		
Part III	4. formalization of learnings	
Chapter 10		all
Chapter 11		

Table 1.1: Outline of this thesis related to ADR stages and research questions

reports on four exploratory case studies (ECSs) in which possible method elements are created to show the feasibility and reflect on the requirements of such a method in order to answer RQ 5, RQ 6, and RQ 7. The chapter titles reflect the choices that will be made in Part I.

ECS 1. Chapter 6: Specifying Microservices

ECS 2. Chapter 7: Using Mockups to Validate Requirements

ECS 3. Chapter 8: Deriving a Normalized System

ECS 4. Chapter 9: Generating Mendix Applications

These exploratory case studies are themselves structured according to the ADR stages as well (see Section 2.2.1).

- The formalization of learnings is done in Part III:
  - In Chapter 10, answers to the research questions and a reflection on the main research challenge are provided, and the contribution towards the method as a whole and the impact of this research in a broader perspective are outlined.
  - The discussion in Chapter 11 contains of a critical reflection on this research and provides directions for future research.

Part I

Theoretical Background

"No action without research; no research without action"

Kurt Lewin

2 Research Approach

The goal of this research is to contribute to the design of a method, including supporting tools, for the development of software from enterprise models that addresses the identified needs, can be applied repeatedly, and is adaptable to specific situations. Due to the uncertainties mentioned in Section 1.2, this research is considered part of the 'fuzzy front-end' [247]. In order to get a better understanding of the problem as well as to get a first idea of possible method elements, a practice-driven approach is adopted, combining theoretical and practical research.

Lewins statement (above) underpins this combined approach, that is common in exploratory research: The only way to meet the goal to have a method, is by doing research – "no action without research" – while, at the same time, this research needs to be nourished and validated by actually changing the world – "no research without action". This chapter will first outline the possible and relevant research approaches for performing a fuzzy front-end research, and then describe the approach that will be applied in this research.

2.1 Possible Research Approaches

In Section 1.3 the main research challenge was formulated and detailed into theoretical and practical research questions. The theoretical research questions require literature study but will require validation in practice. The practical research questions build upon the theoretical research questions and require practical research to provide an answer. Although there are barely universally applicable approaches or best practices to execute an effective fuzzy front-end [373], it often requires several iterations of alternating between creating a possible solution element and applying and evaluating it in practice [134]. This approach is in line with

the earlier observation that methods often result from structuring or generalizing procedures or approaches that are being applied in practice (see Figure 1.4).

Exploring a problem as well as solution space is at the core of (exploratory) case studies [442, 497, 89], an approach that is used often in the fuzzy front-end [432]. An alternative approach to design elements and to test the hypothesis that enterprise models can be used to create adaptable software, is experimentation. Both method elements and the method as a whole can be considered artifacts that need to be designed and evaluated, as described by Design Science Research and Action Research. In this section these different approaches are explored.

2.1.1 Experiments

Experiments seek to test a specific hypothesis through deliberately manipulating the environment [413], and require *a)* explicit manipulation of an independent variable, *b)* random assignment of subjects, *c)* measuring one or more dependent variable(s) – the outcome –, and *d)* to hold constant all other variables. As it is difficult, if not impossible, to isolate variables or even to repeat an intervention with the change of just a single variable in practical settings [226], this approach does not support the creation of a method based on practical research.

2.1.2 Case Studies

In contrast to (field) experiments, the case study approach aims to explore a phenomenon in its natural context [442, 497, 89], with no specific defined outcomes [31]. The case study approach can impose limitations, including lack of rigor and being unable to generalize conclusions [442, 497]. However, by appropriately designing a single case study with its own research approach [442, 497, 89], it can satisfy standards of scientific research [280, 152]. Comparing results of multiple case studies can further impair the limitations [131, 89]. As a result, the (exploratory) case study methodology is a useful research and evaluation tool [497].

2.1.3 Design Science Research

The design science research paradigm is fundamentally a problem-solving paradigm [212, 485], with roots in engineering and the science of the artificial [418]. Design science research aims at constructing and evaluating an artifact designed to meet the identified business need and is therefore complementary to behavioral science that seeks to develop and justify theories that explain or predict phenomena related to the identified business need [301, 150]. Artifacts produced include [301] *a)* conceptual designs, *b)* methods, *c)* models and systems, and *d)* (better) theories.

Because design is inherently an iterative and incremental activity, Hevner suggests three cycles for design science research [211] that can be applied in as many iterations as needed (Figure 2.1):

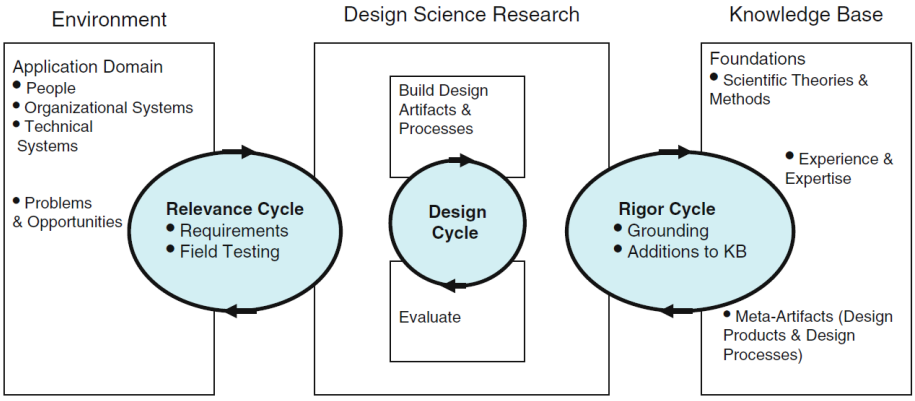


Figure 2.1: Design Science Research Cycles [211]

- The *relevance cycle* provides the requirements for the research and the acceptance criteria for the evaluation of the results. These results determine whether the resulting artifact improves the environment and whether additional iterations are needed.
- The *rigor cycle* provides past knowledge to the project and ensures new contributions are added to the knowledge base. This cycle distinguishes science from (best) practice.
- The *design cycle* where the artifact is constructed and evaluated and (thus) the heart of any design science research project. It takes the requirements from the relevance cycle and theories and methods from the rigor cycle as inputs for both the construction and evaluation phase of the artifact. Often, multiple iterations are needed before contributions can be output into the relevance and rigor cycles.

Peffers suggests a more elaborate stage model that can be applied iteratively [355], but contains more or less the same activities and ideas.

The case study method is considered the most suitable (empirical) method to perform design science research, compared to, e.g., surveys, interviews and experiments [442] and multiple case studies fit well into the Design Science Research iterations [210]. Although performing multiple design iterations can be time-consuming [389], it is considered a good way to design a case study [442]. However, in order to successfully apply Design Science Research, the goal and requirements for the artifact to be created need to be clear [301, 476, 485]. While both the method as a whole and the possible method elements can be considered artifacts to be constructed and evaluated, the requirements for those artifacts are not clear yet. It is expected that only by following the design cycle and evaluating the results *in practice*, the requirements will become more clear.

2.1.4 Action Research

In the 1940s Kurt Lewin introduced the term 'action research' [1] as a research approach in which the action researcher and a client collaborate to solve an organizational problem [63]. It assumes the world to be constantly changing and the research as being part of that change [82]. Action research combines theory and practice and uses a simple, practical, and iterative process to diagnose the problem, intervene with action and reflect on the learnings of the interventions in order to get increasingly better results [29, 18].

Action research consists of a number of activities that are usually applied iteratively (see Figure 2.2):

a) diagnosing and *planning* in order to initiate change;

b) implementing the change (*acting*) and *observing* the process of implementation and its consequences; and

c) *reflecting* on the results and process.

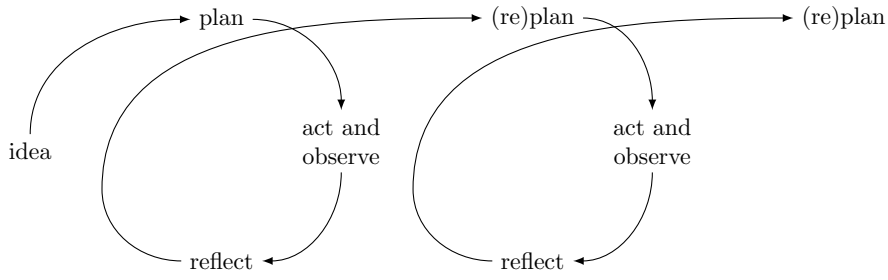


Figure 2.2: Action Research Spiral, adapted from [438, 240]

Action research works well in situations that aim for *a working* solution, rather than *the best* solution. As the researcher is active as an IT consultant, he has access to customer situations where typically time and budget is an important constraint in creating artifacts. Action research allows for the creation of a plan or procedure that can be improved in different iterations by applying it several times on the same or different situations. However, action research suffers from the same disadvantages as single case studies, in the sense that without a proper design they lack the rigor and generalizability [28, 21]. Iterating between and across separate case studies could be a way to circumvent some of these issues [28, 214]. By studying multiple cases, conclusions can be drawn over the similarities and differences between the cases [31, 497].

2.1.5 Action Design Research

ADR is the result of combining design science and action research in order to generate design knowledge through building and evaluating artifacts [409]. It helps researchers to both make scientific contributions and to assist in solving current

and anticipated problems of practitioners [81, 193]. ADR facilitates this approach by having different stages that are mutually influential and can be executed in parallel (see Figure 2.3):

1. problem formulation;
2. building, intervention, and evaluation;
3. reflection and learning; and
4. formalization of learning.

ADR has successfully been applied in the creation of methods [356].

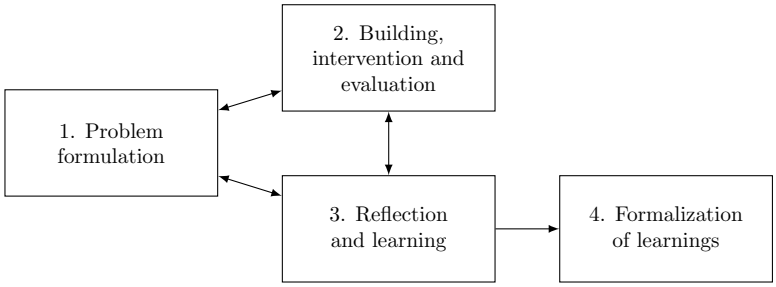


Figure 2.3: Action Design Research stages, adapted from [409]

2.2 Approach as Applied in This Research

The research challenge is to contribute to the design of a method. As methods and method elements often result from structuring or generalizing procedures or approaches that are being applied in practice [199, 206], a practice-driven approach is needed to answer the research questions. Since the research challenge is broad – as is typical for the field of engineering and design science research – and the solution direction has to be developed through interventions in practitioners' problems – as is typical for action research – ADR is specifically applicable for this research. As an IT consultant, the researcher is also a practitioner with access to real-world cases, providing the perfect combination to apply ADR in practice. The

Research Approach	**Applicability**
Experiments	No
Case Studies	Yes
Design Science Research	Yes
Action Research	Yes
Action Design Research	Yes, dominant approach

Table 2.1: Usability of different research approaches for this research

dominant approach for this research will be Action Design Research, combining design science research and action research (see Table 2.1).

In terms of ADR, the problem formulation (stage 1.) for this research has been explored in Section 1.1 and Section 1.2, resulting in an initial set of research questions in Section 1.3. By applying ADR, the relevance of these research questions can easily be validated.

The iterative building, intervention and evaluation (stage 2.) should contribute to building a method. As methods and method elements often result from structuring or generalizing procedures or approaches that are being applied in practical cases, this research should use case studies. Performing a case study has shown to be an effective method in both design science research and action research, and therefore is considered effective for ADR. However, it is suggested to use multiple case studies before drawing general conclusions. In order to further add rigor to each (exploratory) case study, a nested ADR approach (see Figure 2.4) will be applied, as explained in more detail below. In practice, this stage is easily combined with reflection and learning (stage 3.) to guide additional iterations, involving theoretical research, practical research, or a combination of both. By performing multiple iterations, answers from theoretical research can be validated by practical research, while practical research might require additional theoretical research.

In the last stage, formalization of learnings (stage 4.), the lessons learned from the exploratory case studies are extended to a revised set of requirements and a general solution concept for the method to-be. These learnings should provide answers to the research questions (see Table 2.2), in order to reflect on the main research challenge.

RQ 1	Initial answer provided by literature study in Chapter 3, validated by ECSs in Part II
RQ 2	Initial answer provided by literature study in Chapter 4, validated by ECSs in Part II
RQ 3	Initial answer provided by literature study in Chapter 3, validated by ECSs in Part II
RQ 4	Initial answer provided by literature study in Chapter 5, validated by ECSs in Part II
RQ 5	Initial answer provided by literature study in Chapter 4, validated by ECSs in Part II
RQ 6	Initial answer provided by literature study in Section 1.1.2, validated and extended by ECSs in Part II
RQ 7	Initial answer provided by literature study in Chapter 5, validated and extended by ECSs in Part II

Table 2.2: How the research questions are answered by the research approach

2.2.1 Adopting a Nested ADR Approach

As said before, methods and method elements often result from structuring or generalizing procedures or approaches that are being applied in practice. While a specific situation can use and improve a procedure or algorithm to convert an enterprise model into working software by performing multiple iterations, its applicability is likely to increase if a procedure is used and improved by applying it to multiple cases or situations (see Figure 1.4). The creation of one procedure and possible method element, with no specifically defined outcome, is called an *exploratory case study*, with the procedure being the phenomenon to be examined. In order to fully explore the fuzzy front-end of this research, it is important to perform multiple exploratory case studies with different procedures on different organizations. Multiple procedures can, but don't necessarily need to, be evaluated on the same organization.

This research applies a nested ADR approach by considering two levels (see Figure 2.4):

1. the method (research), and
2. the method element (exploratory case studies).

Applying ADR to the method level has been explained in the previous section. In order to further add rigor to the exploratory case studies used in this research, the ADR stages are used to design each of the exploratory case studies, aimed at creating a method element, as well. These exploratory case studies should

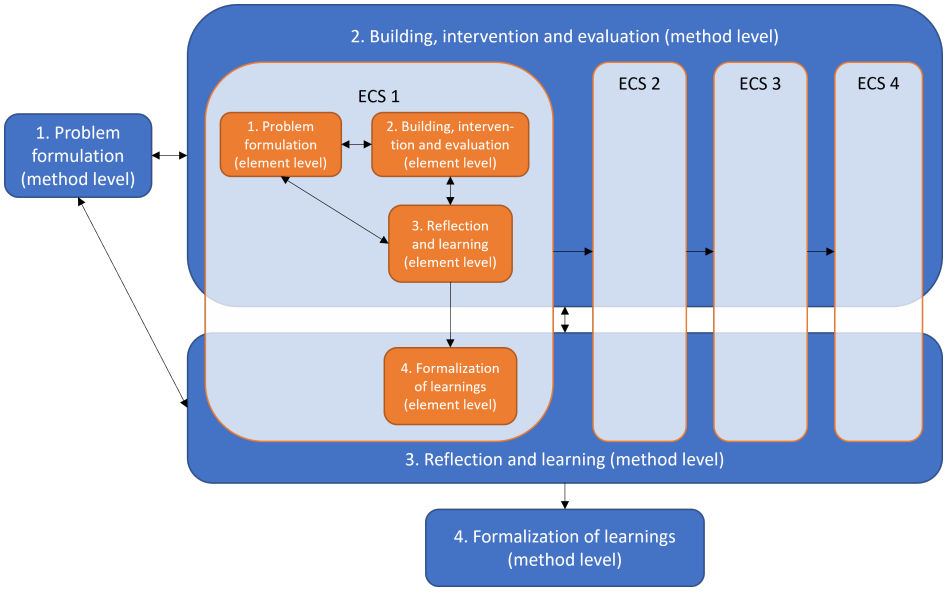


Figure 2.4: Nested ADR approach for this research. All exploratory case studies follow the same process.

not only provide answers to the practical research questions, but also validate the answers to the theoretical research questions. By setting the method element to be created at the heart of each exploratory case study, there is a direct contribution to answer RQ 7. Moreover, they will provide insights in what does – and doesn't – work, contributing to answer RQ 5 and validating initial answers to RQ 1, RQ 2, and RQ 3, as well as provide the opportunity to reflect on the requirements and the needed adaptability for the method, contributing to answer RQ 4 and RQ 6. Applying multiple iterations within each of the exploratory case studies will strengthen the problem formulation for each of the exploratory case studies, and makes sure that the findings on feasibility and usefulness of the created method element(s) are not specific to one case and can be generalized.

2.2.2 Choosing the Exploratory Case Studies

In order to perform exploratory case studies, they need to be selected. As the researcher is also an IT consultant, he has access to real-world enterprises to work with in the creation of possible method elements. By discussing and fine-tuning the problem formulation, the relevance of such a method element, and possibly the entire method, is validated immediately. This approach thus ensures that only elements that are relevant for real-world enterprises are created.

A possible drawback of this approach could be that findings are only relevant for the considered enterprise. When possible, a designed element, being a procedure or algorithm to create software from enterprise models, should be evaluated with multiple enterprises. At the same time, multiple elements can be evaluated

		ECS1	ECS2	ECS3	ECS4
Target technology		Micro-services	Mockups	Normalized Systems	Low code
Needs	1. Complete	X	X		
	2. Evolvability	X	X	X	X
	3. Speed		X	X	X
	4. Traceability	X		X	X
Cases	EU-Rent (academic)				X
	Pizzeria (academic)			X	
	Volley (academic)		X		X
	Patent Granting (IPPO)		three countries		
	Social Housing (ICTU)	X			X
	Subsidy Schemes (anonymized agency)			two schemes	

Table 2.3: Links between the exploratory case studies, the target technologies, the needs addressed, and the cases used for evaluation

against the same set of enterprises when considered relevant. Evaluating multiple elements on multiple enterprises ensures that only elements that are relevant for and applicable to multiple enterprises are created, contributing to the design of a method for the development of software from enterprise models. Table 2.3 summarizes for each exploratory case study, the target technology that will be used (see Section 3.4 for the reasoning of the choice), the identified needs that will be addressed, and the actual (academic and real-world) cases that are used for the evaluation.

"In the computer field, the moment of truth is a running program; all else is prophecy."

Herbert A. Simon [417]

3 Software Development

The quote above tells that developing software (systems) should not be taken light-hearted and that the effort of designing software does not ensure its correctness nor even that it will run. However, this doesn't mean that the design of software can be skipped; they are still systems that should be designed, typically by using models as simplifications of the enterprise and its users and/or the software to-be-developed.

In this chapter an initial answer to RQ 1 and RQ 3 will be provided. As nowadays software development cannot – or should not – be done without making a model, different kind of models will be outlined in Section 3.1 and how they are typically used in the process of software development. As a method needs to be based on a particular Way of Thinking (see Section 1.1.3), in Section 3.2 the General System Development Process (GSDP) is introduced as a generic framework for system design and development that is also grounded in theory and applies a step-wise refinement process. Model-based Engineering (MBE) is an approach towards software development that uses models and knows many flavors, among which is one that addresses need 4; MBE will be explored in Section 3.3. Specific technologies that address needs 2 and 3 are microservices, Normalized Systems, and low-code platforms. As mockups, or user interface prototypes or wireframes, have proven to improve quality and speed in the software development process [374, 378, 25, 148], they will be explored as well. These target 'technologies' are discussed in Section 3.4, and this chapter ends with initial conclusions in Section 3.5.

3.1 Models and Its Relationship With Software

As mentioned in Chapter 1, a model is a simplified representation of a (software) system, that is used to study aspects of the system one is interested in [357, 291, 408, 142, 270, 124] and are key to support human understanding, reasoning and communication [394], allowing for validation before actual implementation [479, 410, 203]. It is therefore considered an important artifact in the software development process [479, 60, 295, 76]. In this section different kinds of models, the notion of ontology, and the typical relationships between models and software code are explored.

3.1.1 Model Kinds

Different kinds of models that are used within the software development process can be distinguished (see Figure 3.1). Guarino et al. distinguish conceptual models from non-conceptual models [186]: non-conceptual models are representations of the real world, including physical (architectural) (scale) models and models representing the inner workings of a (computer) system [186]; conceptual models on the other hand represent the *conceptualizations* of a domain, i.e., it uses abstractions of reality expressed as concepts with clearly-defined semantics [188, 216, 186].[1] This view seems similar to that of NIAM (see Section 4.1.3), that distinguishes conceptual models from physical models [491]. Conceptual modeling is considered a fundamental activity within software development [188].

Seidewitz discerns between descriptive and prescriptive models [408]: a descriptive model describes reality, but reality is not constructed from it; a prescriptive model on the contrary prescribes the structure or behavior of a desired reality which then has to be constructed according to the model. Conceptual models are by definition descriptive, while implementation models are prescriptive [174].

Seemingly related, others discern between conceptual, logical and physical models [380, 501, 474]: conceptual describes what is being done, logical is about how things are done and physical with what specifically. This layering conforms to the types of change as defined in need 2, but this definition of conceptual can but does not necessarily conform to the definition provided earlier. This definition of conceptual however does relate to what is known as 'essential': independent of specific technology; only what is done, not how, when or by whom [309, 480, 305, 499].

Chaudron et al. distinguish three levels of model abstraction [76]:

- *architecture*, defined as a high-level abstraction in terms of components and layers and their relations;
- *design*, defined as a medium abstraction of the implementation of the system, represented in classes and packages and the relations between them; and
- *implementation*, the model that mirrors the actual implementation.

[1] Semantics is used here as a synonym for 'meaning'.

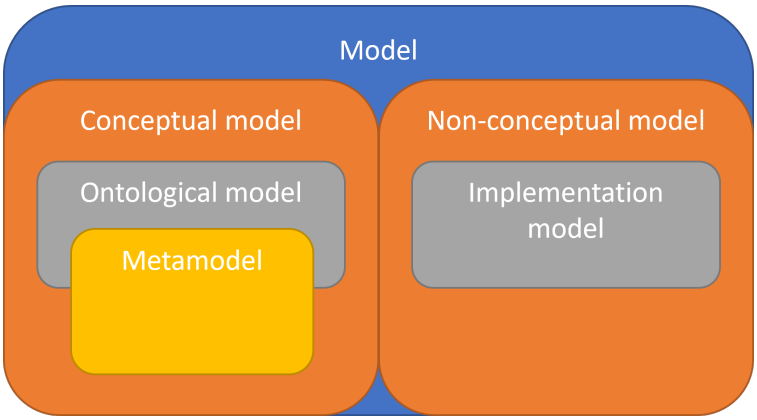


Figure 3.1: Model kinds, expressed in a Venn-diagram

It is clear that the implementation level is of the non-conceptual kind, but for the other levels it is still unclear.

Ontology

An ontology[2] is defined as a formal and explicit conceptual specification describing (shared) knowledge about some domain [185, 53, 436, 188]. An ontology and can be expressed in a model containing the relevant concepts including their definition or semantics, and their relations [157, 427, 187]. An ontology or ontological model is therefore considered a descriptive, conceptual model [13]. In contrast to, e.g., 'regular' (data) models, an ontology specifies a shared view of the involved stakeholders [427].

Metamodels and Modeling Languages

Metamodels are a special kind of models: Theoretically, a metamodel is a conceptual model of a modeling language [57, 237]. A modeling language consists of three components: an abstract syntax, a concrete syntax, and semantics [443, 237, 250, 169]. The abstract syntax provides the constructs and their relationships as well as rules that apply [428]. The abstract syntax can be described by a grammar or metamodel [237, 270, 153, 189, 43, 275]. The concrete syntax provides the notation to express the constructs and relationships of the abstract syntax, and can be, e.g., graphical or textual or a combination [237, 394]. There can be multiple concrete syntaxes that conform to the abstract syntax [250]. The semantics describe the meaning of the concepts in the abstract syntax [250, 52]. A semantics can be formal or informal [394]; the formal semantics can be defined by using an ontology [52].

[2]Etymologically, the word ontology means the study of being and the essence of things, see https://www.etymonline.com/word/ontology.

In practice, metamodels are only capable to express the concrete syntax [236, 169, 65], describing the possible structure of (valid) models, i.e., instances of the metamodel [408, 13, 428, 227], mostly with an implied or informal semantics. However, without a formal semantics, multiple developers can interpret the same model in different ways; only with formal semantics it can be guaranteed that the interpretation conforms precisely with the conceptualization that was modeled [190, 394]. Ontologies have shown to be helpful in reducing semantic ambiguity [114, 452]. The closer a metamodel is to an ontology, the better it can express real-world phenomena [188, 189].

3.1.2 Relationships Between Models and Software

Brown provides a view on the possible relationships between code and models [60] (see Figure 3.2). He uses the term *code-only* for the approach that completely ignores models or uses only informal models. On the other end of the spectrum lies the in practice closely related *model-only* approach where models are used exhaustively to create designs of the software system, but the actual software development is disconnected from the models. Both are considered inadequate for creating and maintaining large-scale or complex software systems [60]. *Code visualization* provides a direct representation of the code, sometimes referred to as implementation model. Although this improves understanding of the system that has been built, it does not aid in designing the system upfront [60]. Brown defines *round-trip engineering* as an approach where formal models and code are constantly and mostly manually being synchronized in order to be able to verify the actual implementation to the designs. As approaches can be combined, the 'flavor' where formal models are being used during software development, but are manually converted to code, can be defined. In the *model-centric* approach, models are the primary deliverables of developers and they are created to such a level of detail that they can automatically be converted to code [60].

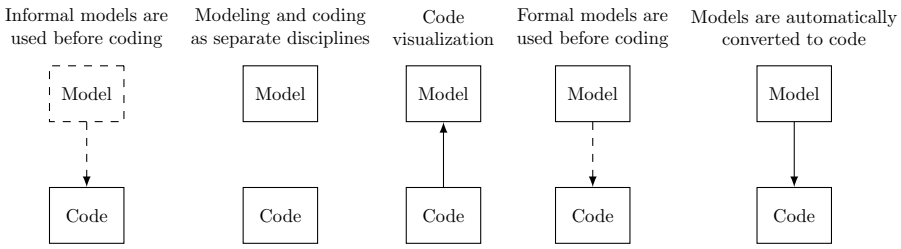


Figure 3.2: Possible relationships between modeling and coding, adapted from [60]

3.2 General System Development Process

System development is the bringing about of a new system or of changes to an existing system [125]. Many frameworks or models exist to understand the

development of systems that may cover the whole or parts of a system's life cycle [390, 365], from inception, planning, analysis, and design, until development, deployment, maintenance, and disposal. In this research the main focus is on the analysis, design and development, and not so much on the other phases. The GSDP is chosen as it addresses exactly those phases, is grounded in theory, and is fully aligned with the Viable System Model (VSM) that helps to design adaptable systems [32, 133, 73, 320].

The GSDP (see Figure 3.3) comprises all activities that have to be performed to arrive at an implemented system, called the Object System (OS), that is part of and used by an environment, called the Using System (US). It distinguishes three phases, i.e., (general) design, technical design, and (actual) implementation, while architecture deals with alignment between systems within a landscape. Both the three phases and the notion of architecture will be explained below.

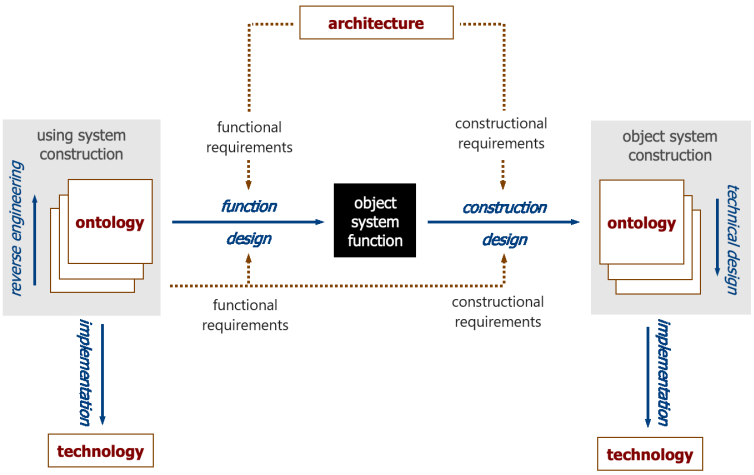


Figure 3.3: The General System Development Process (GSDP), adapted from [120, 125]

3.2.1 System Design

In the GSDP, the system design phase includes *function design* and *construction design*.

Function Design

According to the GSDP, function design starts from the construction of the US and ends with the function of the OS. The result is a functional or black box model clarifying the behavior of the OS in terms of the construction of the US. The input of this activity are *functional requirements*, provided by the US. The ontological model of the US can assist in specifying and validating (the complete set of) requirements and detect unfounded requirements.

Construction Design

In the GSDP, construction design starts with the specified function of the OS and ends with the construction of the OS. The result is a constructional or white box model of the OS containing all constructional specifications for the OS to be built. The input of this activity are *constructional requirements*, provided by the US, also known as *non-functional* requirements. The constructional specifications should be feasible given the available technology, budget and timelines. It could be that this requires compromising or iterating between (reasonable) requirements and (feasible) specifications.

3.2.2 Technical Design

According to the GSDP, technical design is the activity in which a number of subsequently more detailed constructional models are produced. Every subsequent model is derived from the previous one, taking into account the available requirements. A technical design starts from the highest-level (ontological) model that is completely independent of the way in which it is implemented, and ends with the implementation model that fully depends on the applied (abstract) technologies. This view conforms (roughly) to the model kinds as described in Section 3.1.1. Common technologies to use include human beings and IT.

3.2.3 System Implementation

In the GSDP, by implementing a system is understood the assignment of (concrete) technological means to the elements in the implementation model, so that the system can be put into operation. For software, implementing means the actual installation of the software on a device so it can start doing the actual work. For human-beings it is more a matter of telling them to start doing the actual work, possibly by using a device or certain piece of software.

3.2.4 System Architecture

A general challenge in system development is the (too) large amount of design freedom that is left when all (user) requirements are satisfied [124]. Architecture helps designers to use this freedom in a purposeful and systematic way [184, 363]. Most architecture requirements hold for a class of systems instead of for a single system, and are known as *principles* or standards [448, 222]. Applying architecture during systems development balances and safeguards the interests, concerns, and objectives of all stakeholders, broader than just the parties involved in the US and OS [120]. Benefits of architecture include improved integration, reusability, scalability, performance, adaptability, agility and cost savings [329, 440].

3.3 Model-based Engineering

Model-based Engineering (MBE) is an approach towards system development where models are created and analyzed to predict and understand its capabilities [144]. As outlined in Section 1.1.3, nowadays creating – professional – enterprise software without the use of models is unimaginable. As shown in Section 3.1.2, many 'flavors' of MBE can be discerned, ranging from the use of informal models to more formal models, and ranging from more manual to more automated conversion. In fact, automated conversion is only possible from formal models and is better known as Model-driven Development (MDD). In this section the different flavors are explored and it is concluded that Model-driven Software Development (MDSD) best fits the GSDP and answers some of the identified needs.

3.3.1 MBE and Related Notions

Flavors of MBE include Model-driven Engineering (MDE), MDD, and Model-driven Architecture (MDA) [10, 56]. While sometimes these notions are used as synonyms [337, 56], there are differences to distinguish [56, 69]. In fact three different axes can be used to describe the differences: Model-based or model-driven, the scope, and engineering or a part thereof. These axes are detailed below (see also Figure 3.4).[3] Not all possible combinations exist, as summarized in Table 3.1. For example, MDA[4] is defined by the Object Management Group as an example of MDD that uses UML to visualize and (partly) generate code [335] but other uses of Architecture in the field of MBE are not found in literature.

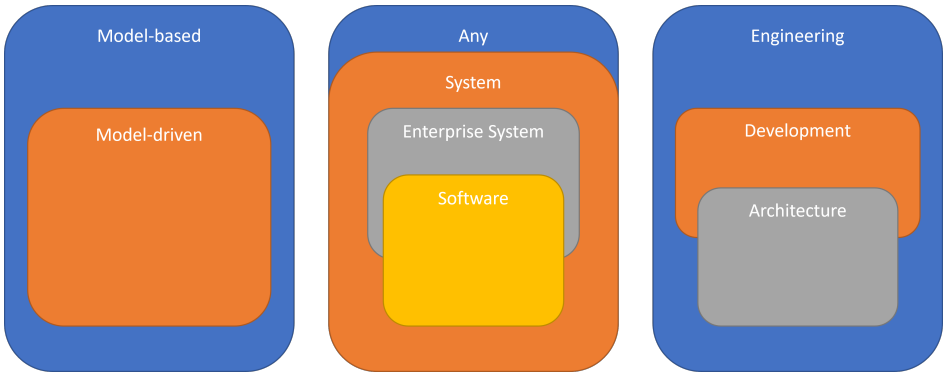


Figure 3.4: Venn diagrams of the different axes in the domain of MBE, adapted from [10, 56]

[3]For more details, see, e.g., https://modelbasedengineering.com/glossary/.
[4]Some say it has nothing to do with architecture [85].

Based	Any (System)	Engineering Development Architecture	MB(S)E [495, 315, 144, 204] MB(S)D [505] MB(S)A [483]
	Enterprise System	Engineering Development Architecture	Not found in literature
	Software	Engineering Development Architecture	MBSE [203, 318, 503] Not found in literature
Driven	Any (System)	Engineering Development Architecture	MD(S)E [241, 39, 403, 160, 462] MD(S)D [313, 410, 23, 160, 56] MD(S)A [41, 312, 251, 335, 56]
	Enterprise System	Engineering Development Architecture	MDESE* Not found in literature
	Software	Engineering Development Architecture	MDSE [462, 285, 56] MDSD [38, 428, 478] Not found in literature

* Only found in call for papers: https://edocconference.org/2022/

Table 3.1: Mentions of MBE and related notions in literature

Model-based or Model-driven

Model-based (MB) uses models in the process, i.e., models play an important role but are not necessarily key artifacts. An example is to manually write code based on models. *Model-driven* (MD) can be seen as a subset of model-based where models play a key role in, hence, 'drive', the development. An example is the automatic generation of code. Usually, this comes with a set of model transformations [412], defined in mapping functions [313]. Going back to Figure 3.2, the most right one is considered a model-driven approach, while the others are model-based [285].

Scope

The scope can be System (S), Enterprise System (ES) or Software (S),[5] or can be left out to indicate any scope. As in the world of engineering everything is considered a system [294], the *System* scope is considered synonymous for any scope. The *Enterprise System* scope is a subset of System, aimed at enterprises, including their ISs. Software is a subset of System, focused on software. Software can be but not necessarily is an Enterprise System and v.v.

[5]Indeed, S can refer to both System and Software. In this thesis S is used only for Software specifically.

Engineering or Part Thereof

Engineering (E) in the context of MBE refers to the process of applying (scientific) engineering principles to the design, development and maintenance of structures, machines, processes, etc. Development (D) in the context of MBE is seen as part of engineering process in which, often in multiple steps or iterations, an artifact is being produced.[6] Architecture (A) in the context of MBE refers to a part of the engineering process, that could but not necessarily is part of Development.

3.3.2 Model-driven Software Development

MDD or, more specifically, MDSD puts models at the heart of software development with the promise to bridge the gap between requirements of an organization and the technical implementation of the software system by means of model transformation [412, 315, 428, 352, 43, 56], using the models to generate code or for real-time interpretation by running software [149, 203, 56]. While there are differences between code generation and model interpretation in terms of easiness to understand, debugging possibilities, performance of the execution environment, and compile and deploy time, from a usage perspective these differences don't really matter; both need a mapping (or transformation definition) from higher order (business domain) model to lower order software model, and they can even be combined [68, 113, 56]. Sometimes this mapping is called 'forward engineering' to explicitly address that the process moves from a high-level implementation-independent abstraction to a physical implementation of the (software) system [141], fully compliant with the GSDP (see Section 3.2).

The first mentions of a model-driven approach are from the '90s [259, 132], and traces can be found in the CASE tools [160, 292] that were based on SADT [384, 500, 109] and SSD [154] (see also Section 1.1.3). In a way, MDD goes back to introducing Assembly language[7] [51] as an abstraction over machine code[8] [192], after which many more abstractions followed, including Third-Generation Programming Languages (3GLs) such as FORTRAN, Cobol, and object-oriented programming languages such as `C++` and Java.

MDD hides complexity of its underlying technologies by raising the level of abstraction [192, 160, 285] and therefore offers a promising approach to address the inability of 3GLs to alleviate the complexity of software development [403]. The promise of Fourth-Generation Programming Languages (4GLs) is that non-programmers can create applications, including data management, report generation and web development [304]. Current implementations of 4GLs include Uniface,[9] SAP PowerBuilder,[10] and basically all low-code platforms (see Section 3.4.4). As it raises the abstraction level to the business, 4GLs can be consid-

[6] This does seem to conform to the definition of technical design as defined in the GSDP (see Section 3.2.2).

[7] Known as Second-Generation Programming Language (2GL), see, e.g., `https://www.ibm.com/docs/en/zos/2.1.0?topic=hlasm-language-reference`.

[8] Known as First-Generation Programming Language (1GL).

[9] `https://www.rocketsoftware.com/products/uniface-application-development-platform`

[10] Currently known as Appeon: `https://www.appeon.com/`.

ered a subset of Domain Specific Languages (DSLs) [463, 202]. Currently, 3GL and 4GL co-exist [11].

Moreover, Fifth-Generation Programming Languages (5GLs) have been defined as logic-based programming languages that use constraints to solve problems rather than (user defined) defined algorithms [273, 228]. Prolog[11] is one of the few implementations of 5GL.[12] Currently, 5GLs are considered too limited to create enterprise software [429]. Thalheim et al. define Modeling-as-Programming as a necessary first step towards real 5GLs [446, 445].

MDD can start from a DSL or from a more generic modeling language [55]. MDD is currently also known as model-as-code [430, 347] and therefore closely related to trends such as infrastructure-as-code [319], and everything-as-code [433].

Model Transformation

Metamodels play a crucial role in defining a model transformation or mapping function [13, 43, 26, 234] (see Figure 3.5) and need to be precisely defined [315, 248, 56]. A model transformation definition or mapping function has its root in compiler construction [4][13] and basically consists of three parts [453, 43]:

1. identify elements in the source (meta)model[14] that need to be transformed;
2. for each of the identified elements produce the associated target element(s) (in terms of the target metamodel); and
3. produce the tracing information that links target and source elements.

A model transformation or mapping function can be defined in various ways, imperative or declarative and using graphs and/or rules, and can be defined as a precise procedure, algorithm or mathematical function [90, 43, 248] and thus allows for formal reasoning and verification.

Advantages of MDSD

Main advantages of an MDSD approach include *a)* a common and better understanding and reasoning about the required or created system [26, 203, 56], *b)* the possibility to simulate before building the system [248, 203], *c)* an increased productivity of the development team due to (partial) automation of the development process [248, 26, 203, 56], *d)* a reduction of the number of bugs or defects, as they can be discovered early in the development process when they are less costly to fix [248, 203, 56], and *e)* traceability between model and the code [5, 396].

[11] https://www.iso.org/standard/21413.html.

[12] ChatGPT can be considered an implementation towards 5GL, see https://dev.to/peibolsang/conversational-programming-4153.

[13] On the lowest level, creating machine code that can be executed by the Central Processing Unit (CPU) from a higher level programming language is known as a compiler [410, 43].

[14] This can be multiple models that, for simplicity, together are considered as one model.

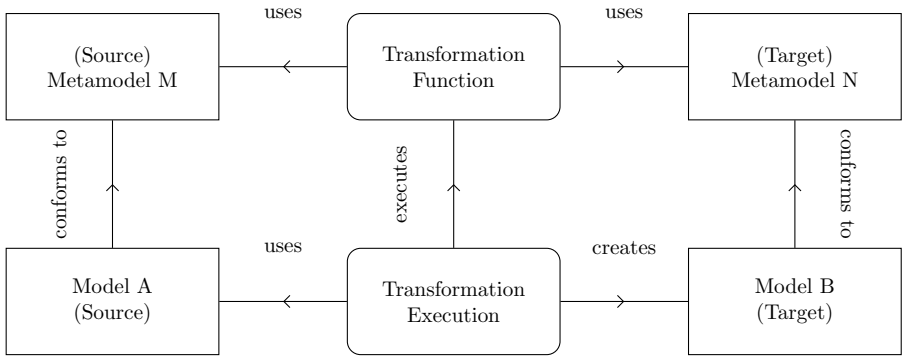


Figure 3.5: Model transformation and the role of metamodels, adapted from [40, 90, 43, 56]

Disadvantages of MDSD

Reported drawbacks of MDSD include: *a)* it is considered complex [9, 192, 403, 160, 285]; *b)* it is hard to include custom code [160]; *c)* it tends to leave user interface (UI) aspects to the end of the development [377]; *d)* it moves complexity to the models instead of the code [192]; *e)* it is not considered agile or flexible [312]; *f)* it needs input models with formal semantics [192, 236, 403, 160, 394].

Regarding disadvantage *a)*: This is not solved easily. However, due to advancements in technologies, the step from (enterprise) models to a software model is becoming more and more easy.

Regarding disadvantage *b)*: Technologies such as low code have shown that it is now easier to combine code generation and insertion of custom code. Low-code platforms are explored in Section 3.4.

Regarding disadvantage *c)*: Traditionally, mockup interfaces are used to check the usability before creating the software. Technological advancements such as low code have made it possible to create interesting and usable user interfaces from the start of software development. Both technologies will be explored in Section 3.4.

Regarding disadvantage *d)*: Although this can be considered a disadvantage, for this research it is considered the (only) right approach. Most complexity in software comes from the enterprise level [177] and thus should be made visible at that level by means of enterprise models.

Regarding disadvantage *e)*: The reasoning is that first a model must be created, and then there still is the effort to create software. However, when the models are executable or can be converted to code automatically, they can be seen as if they *are* the code. In that case, the same agile principles that apply to programming apply equally well to modeling [311, 285]; such an approach can only enhance the agility, especially on the enterprise level.

Regarding disadvantage *f)*: in this research this is not considered as a dis-

advantage, as in fact every model used in software development should provide semantics (see Section 3.1.1); it is however a concern that implies the need to carefully choose a modeling language with a formal semantics.

Usability for This Research

From all the flavors of MBE, it is concluded that the best fit for this research is a model-driven approach towards the development of high quality enterprise software that starts from enterprise models. For the remainder of this thesis, this will be called Enterprise Model-driven Software Development (EMDSD), as a specialization of MDSD: *Enterprise Model* as the approach starts from the model of an enterprise; *Model-driven* as this approach needs to bridge the gap from enterprise model towards software while automating this step ensures speed (need 3) and traceability (need 4); *Software* as the approach is focused on creating software for enterprises;[15] and *Development* as the goal is to create working software in one or more steps or iterations, in line with the technical design part of the GSDP. While MDSD addresses some of the identified needs, for needs 1 and 2 it relies on the quality of the input models and the evolvability of the output technologies respectively.

Applying MDSD implies defining a model transformation or mapping function from the input model(s) to the target technologies. For the input models an existing DSL could be chosen. Downsides of using a DSL is that they are hard to develop [316, 255] and domain-specific and thus only usable within a specific domain [439]. As the method to be designed should be universally applicable, a more generic (enterprise) modeling language should be chosen. Such a modeling language, preferably based on a Way of Thinking, should also help in describing all business requirements, addressing need 1. MDSD indeed requires that the model describes all business requirements, is free from contradictions, and has its metamodel and semantics formally and precisely specified [410, 198, 192, 56] (see also Section 3.1.1). These are considered as criteria in choosing the (input) enterprise modeling language in Chapter 4.

In order to further address the need 2, the kind of changes in the enterprise that should be supported by the generated software need to be made explicitly. In order to further support this need and in order to deal with the issue of incorporating custom code (see disadvantage *b)* in Section 3.3.2), target technologies that support evolvability will be selected in the next section.

3.4 Target Technologies

While MDSD addresses the needs 3 and 4, and it offers a way to regenerate the system with new input models and therefore provides some flexibility (need 2), concerns of MDSD include a lack of integration of custom code and poor user interface (UI). Therefore, in this section, some modern technologies that are built

[15] The focus of this research actually is on Enterprise Software, the overlapping part of Software and Enterprise Systems in Figure 3.4.

around offering flexibility and/or a good UI, and sometimes even for speed, are explored; mockups for speed and good UI, microservices for flexibility, normalized systems for flexibility (and speed), and low code for all of them. As MDSD requires a model transformation, the metamodels are provided where possible.

3.4.1 Mockups

Mockups, also known as user interface (UI) prototypes or wireframes, have become a very popular artifact to capture requirements and to evaluate the usability of software before it is actually created [332, 377]. A mockup is a sketch or visual prototype of a possible UI of the application that helps to agree on broad aspects of the UI [377]. A mockup can be considered as a model of the software to-be-developed, providing a way to get early feedback on both the requirements and the implementation [74, 300]. The main advantage of a mockup is that it is understandable by both end users and developers [323, 300]. Using mockups in the software development process has proven to improve quality and speed [374, 378, 25, 148].

Many mockup tools currently exist, including Axure,[16] Balsamiq,[17] Figma,[18] Marvel,[19] Mockflow,[20] Mockingbird,[21] Moqups,[22] Pencil,[23] Sketch,[24] and Wireframe.[25] Disadvantages of mockups include that they are usually thrown away after development, and that they have to be converted manually into working software [145]. It has been shown that mockups can be used in an MDSD approach [377, 37, 351], which is a way to overcome the aforementioned disadvantage. Low-code platforms (see Section 3.4.4) can be considered advanced mockup tools.

3.4.2 Microservices

Microservices are an architectural and organizational approach to software development where software is composed of small, independent services that communicate with each other through Application Programming Interfaces (APIs) [328, 156, 492]. Microservices are independently deployable, operable and scalable and may be implemented with different technologies [155, 115]. In practice these services are owned by small, self-contained teams, and typically built around business capabilities [155].

The microservice architecture is becoming a popular market standard, based on service orientation, that aims at improving adaptivity by decoupling (software) systems [156, 129, 111, 289]. It allows organizations to *a)* easier scale applications, *b)* faster develop or change applications independently, while maintaining their

[16] https://www.axure.com/
[17] https://balsamiq.com/
[18] https://www.figma.com/
[19] https://marvelapp.com/
[20] https://www.mockflow.com/
[21] https://gomockingbird.com/
[22] https://moqups.com/
[23] http://pencil.evolus.vn/
[24] https://www.sketch.com/
[25] https://wireframe.cc/

interoperability, *c)* enable innovation, and *d)* accelerate time-to-market for new or changed products and thus increase agility [274, 492, 129, 289].

An issue with microservice is that there is no agreement on the 'right' size of a microservice [492, 225]:

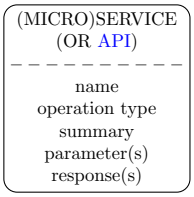
- Steghuis' research for optimal service granularity [431] found mainly functional considerations such as business process flexibility, a maintainable and low-cost landscape, and performance. She recommends splitting services into logical parts with different stability characteristics – leaving open how to discern these "logical parts" and characteristics of stability.
- Compared to "traditional" Service Oriented Architecture (SOA), microservices are (more) fine-grained and protocols of the involved APIs are (more) lightweight [508, 129, 139].
- The largest sizes reported follow Amazon's notion of the Two Pizza Team, i.e., the whole team can be fed by two pizzas, or no more than a dozen people. On the smaller size scale there are setups where a team of half-a-dozen would support half-a-dozen services [155].
- A microservice should not be as small as possible, but as small as needed to make it understandable [140].
- When a service is too big, it should be split into two or more services [129]. This however still doesn't provide an objective measure when a service is too big.
- Moreover, the size of a microservice might depend on the business and organizational context [509] and it is considered bad practice to make the service too small, as the run-time overhead and the operational complexity can overwhelm the benefits of the approach [424, 7].
- Others mention that a microservice should be focused on one specific task [156], one single business capability [129], or one atomic business activity [447]. However, this still doesn't provide a size for a microservice.

Other challenges of applying a microservice architecture include the difficulty to keep data consistent across the enterprise [425], to monitor the resources [225, 425], and to align the organizational coordination in order to manage all the services [129, 225]. However, several solutions to overcome these issues have also been proposed [425].

Application Programming Interface

In order for a (micro)service to be able to be used, it will need to expose an API [22, 66]: a clearly defined method and protocol for communication that define how other services can access the service's functions and data, making it possible to use the service without knowing its internal construction or technology behind it [233]. APIs can be specified by using, e.g., OpenAPI Specification (OAS), a technology independent and widely adopted industry standard for describing HTTP APIs, based on YAML [288] (see Figure 3.6b for an example). Such a

specification describes what a (micro)service does and how it can be accessed, but not how it is actually implemented. According to OAS, an API definition must contain a name, type,[26] summary, response, and, optionally, one or more parameters (see Figure 3.6a). Tools that support this standard[27] can (often) easily generate mock APIs or stubs from these definitions and generate documentation and tests.



(a) (simplified) API metamodel according to OAS [288]

```
paths:
  /users:                                    <-name
    get:                                     <-operation type
      summary: Returns a list of users.      <-summary
      description: Optional extended description in CommonMark or
            HTML
      parameters:                            <-parameters (empty)
      responses:                             <-responses
        '200':
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

(b) Example that defines the `GET /user` endpoint, (micro)service, or API, as a path in OAS, annotated with properties from the API metamodel.

Figure 3.6: (Micro)service metamodel and example

3.4.3 Normalized Systems

Lehman's software evolution laws [281] stipulate that

1) "A program [...] must be continually adapted else it becomes progressively less satisfactory", and

2) "As a program is evolved its complexity increases unless work is done to maintain or reduce it".

On one hand this means that software systems should always change in order to be effective, while on the other hand it means that change will become more difficult over time, unless the software system is designed to be evolvable.

Normalized Systems Theory states that software should not have instabilities, i.e., a bounded amount of additional requirements cannot lead to an unbounded amount of additional (versions of) software primitives [298]. If a change requires increasing effort as the software system grows, this is considered an instability and this is called a *combinatorial effect*. Normalized Systems are software systems that exhibit stability with respect to an anticipated set of changes and, therefore, avoid the occurrence of combinatorial effects and thus accommodate (endless) change [297]. Normalized Systems Theory defines four theorems and five elements to achieve stable yet evolvable software systems.

[26] Typical HTTP operation types or methods include GET, PUT, POST, DELETE, PATCH, OPTIONS, HEAD, TRACE and CONNECT [146, 395].

[27] See `https://openapi.tools/`.

Theorems

Normalized Systems Theory formulates four design theorems (including proofs) for the implementation of stable software systems that can be changed in such a way the impact of the change only depends on the change itself and not on the size of the system as a whole [297, 298]:

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

Theorem 2. *Data version transparency* implies that data should be communicated between components in a version transparent way;

Theorem 3. *Action version transparency* implies that an active component can be upgraded without impacting the calling components, i.e., in a version transparent way; and

Theorem 4. *Separation of states* implies that active components should exhibit state keeping, e.g., to support (stateful) workflow.

Normalized Systems theorems 2, 3, and 4 can be considered special cases of separation of concerns (theorem 1).

Elements

Systematically applying the Normalized System (NS) theorems will lead to a very fine-grained modular structure [298], in a way similar to that of microservices (see Section 3.4.2). As applying these theorems in a disciplined way may ask a lot from software developers, Normalized Systems Theory proposes a set of higher-level software elements that encapsulate software primitives: *data element, task element, connector element, flow element,* and *trigger element*. These elements are modular structures that adhere to the design theorems, in order to provide the required stability with respect to the anticipated changes [297], and are independent of a specific technology environment. Both data and action elements can contain cross-cutting concerns, or *generic supporting tasks*. The metamodel of the NS elements is shown in Figure 3.7. The internal structure of every element can be described by a design pattern that is executable and can be expanded automatically, enabling the automation of software development [297]. The NS expanders, that use *descriptor files*, representing the elements, as input, can be considered an implementation of MDSD [99].

Data Element A data element represents an encapsulated data construct for storing and providing data. It contains various attributes or fields with its get- and set-methods to provide access to the data in a data version transparent way. Generic supporting tasks, also called cross-cutting concerns, such as access control and persistency, should be added to the element in separate constructs [297]. An attribute can either be a link to another data element or a primitive type such as

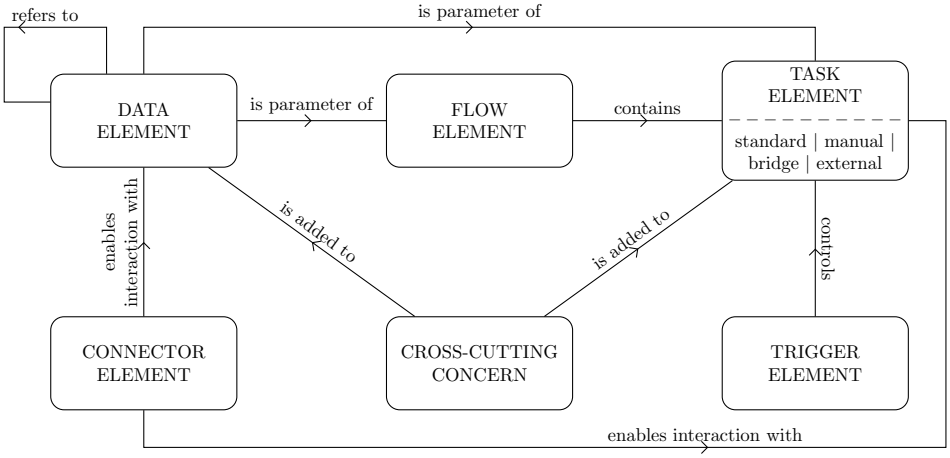


Figure 3.7: Normalized Systems metamodel, adapted from [130]

Integer, String, Boolean, or Date. NS data elements can be visualized by means of an ERD.[28] An example data element is PersonDetails that holds the name and birthday of a person.

Task Element A task element contains a task or action representing one and only one *specific functional* task.[29] Arguments and parameters need to be encapsulated as separate data elements, and cross-cutting concerns such as logging and remote access should be added as separate constructs. Four different implementations of a task element are distinguished:

- In a *standard* task, the actual task is programmed in the task element and performed by the same system.
- In a *manual* task, a human act is required to fulfill the task. The user then has to set the state of the life cycle data element through a user interface, after completion of the task.
- A process step can also require more complex behavior, involving aspects that are not the concern of that particular flow. A *bridge* task creates an instance of another data element that goes through its own designated flow. The data element instances that are created by a bridge task – its 'children' – must report its status to the instance it was created by, making it possible for the creating instance to wait for some state of its child(ren) to continue.
- When another (external) system has been implemented to perform a certain action, the task element can be implemented as an *external* task, calling the other system and setting its state depending on the answer from the external system.

[28] See, e.g., http://en.wikipedia.org/wiki/Entity-relationship_model.
[29] It is up to the developer to decide on the granularity of such a functional task.

An example task element is a PersonFileWriter that can write person data (PersonDetails) to some file.

Connector Element The connector element enables users as well as external applications systems to interact with data and task elements. For every data and task element one or more connector element(s) can be fully and deterministically derived. An example of a connector element is the PersonDetailDeleter that can be used (by other applications) to delete the data of some person.

Flow Element A flow element orchestrates the flow of tasks and contains a sequence of actions and intermediate states that can be described by a state transition diagram.[30] Several data elements are tied to it to keep the state and the history of the state. An example flow element is an OrderEngine that handles the Order from initiation to delivery.

Trigger Element The trigger element provides the functionality to periodically invoke flow orchestration. It controls both the regular and error states and checks whether a task element has to be triggered. An example is a DailyTimerHandler that starts all tasks to create daily reports.

3.4.4 Low Code

The term low code was introduced by Forrester in 2014 [375]. Although some say it is neither clear what low code exactly is, nor that its features are very new [69, 50], low-code development platforms claim to enable creation of software with less effort compared to traditional[31] programming [481]. It builds upon existing concepts such as MDD, code generation and visual programming [116]; most platforms support the generation of data models from importing existing spreadsheets or database and the generation of interfaces (both screens and APIs) for data management, that can be further extended to enrich the user experience, for both desktop, tablet and mobile devices. Depending on the features of the platform being used and the overall system requirements, the developer may or may not have to augment the design with some good old-fashioned code, or the platform may produce an entire working solution with no additional code required [372]. Some better known low-code platforms include Appian,[32] Mendix,[33] Microsoft PowerApps,[34] Oracle APEX,[35] OutSystems,[36] Pega,[37] Quickbase,[38]

[30] See, e.g., https://en.wikipedia.org/wiki/State_diagram.
[31] Also known as 'high code', such as Java and .Net.
[32] https://appian.com/
[33] https://mendix.com/
[34] https://powerapps.microsoft.com/en-us/
[35] https://apex.oracle.com/en/
[36] https://www.outsystems.com/
[37] https://www.pega.com/
[38] https://www.quickbase.com/

Salesforce Lightning,[39] ServiceNow Now Platform,[40] and Zoho Creator.[41]

Claimed benefits of low code include [208, 393, 293, 381, 416] *a)* less hand–coding, faster development, and, as a result, cost reduction in both development and maintenance and a shorter time-to-market; *b)* complexity reduction by using prebuilt components; *c)* the ability for non-technical people to create applications, thus opening up the possible population for application development as well as improving business and IT collaboration while increasing IT productivity; and *d)* enabling digital transformation and increasing IT and business agility.

Low-code platforms support technical variability such as changing from one database to another, or from one front-end framework to another, and they support (quick) implementation of new features as well. And although low-code platforms may (implicitly) adhere to (some of) the NS design theorems, this is not guaranteed. Moreover, these platforms also do not prevent developers from hard coding (implicit) organizational design choices [494, 458] (see Section 4.3).

Other limitations include poor scalability and performance, especially in working with large data sets, vendor lock-in and limited creativity and customizability [416]. Regarding scalability and performance: there is also research that argues otherwise [376, 208]. Low code should not be considered to be the solution for all kinds of software development, but it does support a (possibly wide) range of software solutions. Regarding vendor lock-in: Some platforms offer possibilities to export the models or generated code [167]. It is questionable whether these exports are useful without the tool that created these exports, but on the other hand vendor lock-in also seems to work for some types of customers [506]. Vendor lock-in also isn't tied to low code only, it is quite common within the software branch. When applying an MDSD approach towards low code, the mapping can be recreated for another target platform so that all applications can be regenerated towards another technology. The latter limitation, limited creativity and customizability, can be considered both a pro and a con: by limiting the design freedom for developers, they can focus on what really matters, namely the business impact. On the other hand, once you run into a limitation of the platform, it is hard to work around this. Depending on the platform, it might be possible to insert custom code. Again, it is mainly about choosing the right platform for the purpose; sometimes low code just isn't the right answer.

Metamodel

Low-code platforms rely on three basic concepts (see Figure 3.8): *data*, *logic*, also called action or (micro)service, and *interface*, i.e., API or screen, as well as their interrelations and *permission* (or access control) *rules* for *user roles* to allow users with certain roles to use these parts of the application. Compared to traditional programming languages like Java and .Net, that mainly rely on the concepts of class and method, the metamodel of low code is richer and closer to the business, making it easier to, e.g., connect the information need of a user to an interface to

[39] https://www.salesforce.com/eu/products/platform/lightning/
[40] https://www.servicenow.com/now-platform.html
[41] https://www.zoho.com/creator/

retrieve and show the corresponding data.

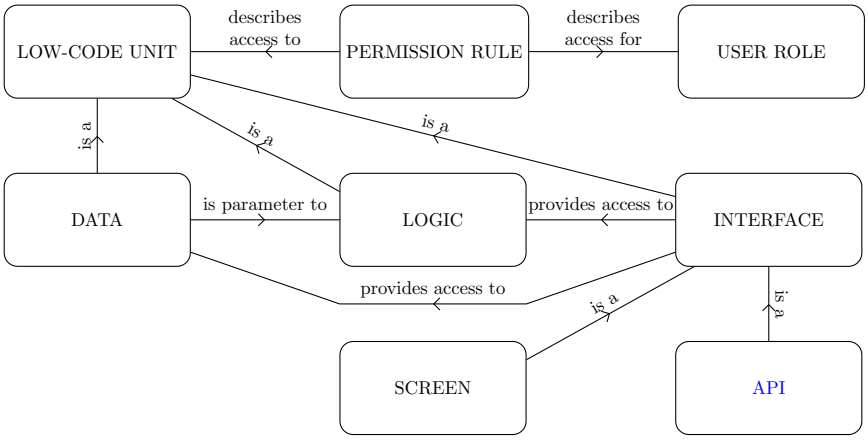


Figure 3.8: Low-code metamodel

3.5 Conclusions

In this chapter the concept of a model is explored as a means to study aspects of a (software) system before it is actually implemented; although there are different possible relationships between models and code, models are considered key artifacts in software development, also known as MBE. In an initial answer to RQ 1, the GSDP explains that (software) system development comprises the activity of detailing high level models, called ontological models, into more specific models, called implementation models. MDSD is a 'nephew' of MBE where models are being used to generate working software by means of model transformation that enables traceability from requirements to implementation, addressing needs 3 and 4. In order to define such a model transformation, the metamodel of both the input and output models needs to be precisely defined. Moreover, the semantics of the input (meta)model needs to be formally defined.

In order to further address the needs, four target 'technologies' are selected, in an initial answer to RQ 3: Mockups improves quality and speed of the software development process, answering the general need for quality and need 3 specifically. Microservices improve agility, answering need 2. Normalized Systems is a theory about system stability, allowing for controlled evolvability, answering need 2. Low code is built around the concept of MDSD for speed and adaptability, answering needs 3 and 2.

Now that the general approach and target technologies to answer the needs are defined, the enterprise models that will be used as input for the EMDSD approach can be identified in Chapter 4. Additionally, identified method elements will be summarized in Chapter 5 in order to provide guidance to the exploratory case studies in Part II.

"Het is gemakkelijk flexibel te zijn, als je geen ruggengraat hebt."
"It is easy to be flexible without a backbone."

Nederlands tegeltjeswijsheid / Dutch saying

4 Enterprise Modeling

This chapter is about Enterprise Modeling in general and Enterprise Ontology and Enterprise Implementation specifically. Building upon need 1 and the specific needs from MDSD, i.e., that the input (enterprise) models need to be precise and have a formal semantics (see Section 3.3.2), in Section 4.1 different modeling techniques for enterprises are explored and criteria in order to select one for the purpose of this research are formulated. The theoretical foundations of Enterprise Ontology and the chosen modeling technique are detailed in Section 4.2.

As explained in Section 1.1, enterprises nowadays need to be flexible in multiple aspects. However, the Dutch saying above implies an enterprise – and its supporting software – cannot be flexible in all aspects; there has to be some stable part or backbone. Enterprise Ontology defines the (relatively) stable parts of an enterprise. As Enterprise Ontology abstracts from implementation, the concept of Enterprise Implementation is explored. The theoretical framework of Enterprise Implementation is created while performing the exploratory case studies, and thus is a key deliverable in this research that will be shown in Section 4.3. This chapter ends with conclusions and an initial answer to RQ 2 in Section 4.4.

4.1 Choosing an Enterprise Modeling Technique

Enterprise modeling is the structured activity in which an integrated and commonly shared model of an enterprise is created [434, 394], preferably supported by practical, flexible and adaptable procedures, tools, best practices [158, 394]. SADT, DFD and ERDs from the 1970s (see Section 1.1.3) can be considered as the first enterprise modeling techniques [475]. An enterprise model can contain multiple aspects highlighting different perspectives [434, 394] and is sometimes considered synonymous to enterprise architecture [165]. Enterprise modeling can

support, e.g., *a)* the understanding of what an enterprise is and does, *b)* the design and analysis of an enterprise, *c)* identifying improvement possibilities for an enterprise, *d)* (strategic) decision-making of an enterprise, and *e)* (re)designing supporting software for an enterprise [475, 394]. Enterprise modeling is an important area within Enterprise Engineering, the body of knowledge containing principles, methods and best practices for aligned design and development of an enterprise [173, 394].

Participative Enterprise Modeling is a way to improve the quality of the models [434, 435] and therefore can be an answer to need 1. As participative enterprise modeling requires the modeling technique to be easily understood by the audience [434, 435], it often means it is not as rigid or formal as EMDSD requires, and therefore this branch of enterprise modeling is further explored.

In this section the different perspectives in enterprise modeling (Section 4.1.1) and the criteria to choose a modeling technique for the purpose of this research (Section 4.1.2) are explored, and several enterprise modeling techniques are outlined (Section 4.1.3) in order to choose one to use in the EMDSD approach as adopted in this research.

4.1.1 Enterprise Modeling Perspectives

Within enterprise modeling, two dominant approaches or perspectives can be discerned: business process centered and data (or object or information[1]) centered [475]. Fox suggests that integrating different perspectives is necessary in order to achieve agility [158]. UML is an example of an enterprise modeling language that combines both processes and data [303, 182].

Others suggest that combining process descriptions and business rules has benefits [512, 322, 510], especially in achieving (process) flexibility [181, 464]. At the same time, business rules and facts or objects are closely related [386, 385, 512, 394]. CogNIAM is a method for enterprise modeling that combines business processes, business rules and data. Using flowcharts[2] is another example [504] that basically consists of process steps, decisions (rules) and (input or output) data. Organizing around services and/or products, defined as a bundle of services, is another trend in enterprises and thus in enterprise modeling [246, 405, 394, 230]. The relationships between these different enterprise modeling perspectives are summarized in Figure 4.1.

The field of Enterprise Engineering sees enterprises as complex sociotechnical systems [173] and therefore connects the fields of organizational science and software engineering. With this view, both actors (human-beings) and software are considered important perspectives as well.

In order to be able to create software *from* enterprise models, the software perspective is left out to keep focus on the other perspectives. Similar research has shown that bridging the gap from these perspectives to software is possible [507]. The perspectives will be reflected in the criteria in the next section.

[1] Data, objects and information are not exactly the same but they are closely related and often considered as being the same approach/perspective.

[2] https://en.wikipedia.org/wiki/Flowchart

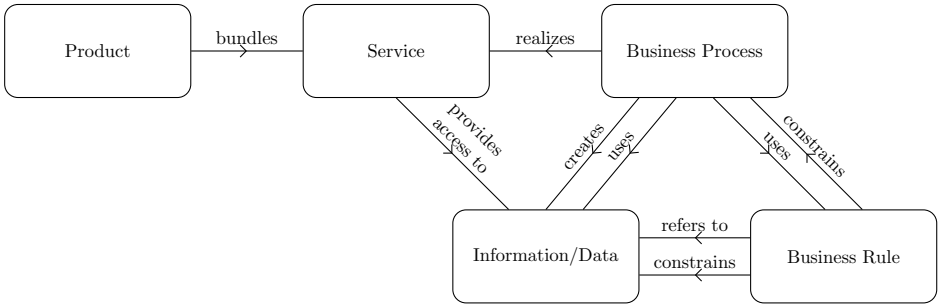


Figure 4.1: Relationships between different enterprise modeling perspectives (business) products, (business) services, (business) processes, (business) rules and information or data

4.1.2 Criteria

In choosing an enterprise modeling technique, consisting of at least a modeling language and modeling procedure [237] (see Figure 1.3), several criteria are defined, partly from literature and partly from experience. A critical reflection on the choice for modeling technique as well as these criteria will follow in the conclusions of this research.

The criteria for selecting an enterprise modeling technique to support the automated creation of software are as follows:

a) the modeling technique is a proper technique, i.e., it comes with a modeling language (WoM) and modeling procedure (WoW);

b) the modeling technique is grounded in theory that provides the way to look at the world (WoT);

c) there are tools (WoS) that support the modeling technique, or, at minimum, the modeling language, so that the models as input for the MDSD approach as adopted in this research can indeed be created (easily);

d) the modeling technique supports the perspectives Product and/or Service, Actors (or Roles), Process, Information and/or Data, and Business Rules (see Section 4.1.1) as they are considered important to create enterprise software;

e) the modeling language of the modeling technique has a formalized abstract syntax, which is required to be able to perform a model transformation (see Section 3.1.1 and Section 3.3.2);

f) the modeling language of the modeling technique has a formal semantics, as this is required to provide unambiguous specifications [52] and to be able to perform a transformation to code *without loss of semantics* (see Section 3.1.1 and Section 3.3.2);

g) there is a good number of example cases that have applied the modeling technique so that the EMDSD approach as adopted in this research can be evaluated on these models;

h) there is an active and good-sized community using the modeling technique that might be able to take on the EMDSD approach as adopted in this research in the future.

Compared to Figure 1.3 the Way of Controlling is left out. The WoC is about managing a development activity or project and includes, e.g., planning, monitoring and evaluating [411, 488, 487]. While it is desirable to have a WoC that is aligned with the modeling technique, there are many existing WoCs such as PRINCE2,[3] The Open Group Architecture Framework (TOGAF),[4] and Agile methods,[5] that are often separated from modeling techniques. At this stage of the research the researcher suggests applying an existing WoC.

In choosing a modeling technique, different techniques can be combined in order to cover all or most criteria. However, at this stage of the research it is considered better, if possible, to choose one modeling technique that covers all or most criteria.

4.1.3 Enterprise Modeling Techniques

In this section the (current) enterprise modeling techniques that can be found in the public domain, i.e., not proprietary are briefly summarized. The focus is on modeling techniques and generic frameworks such as TOGAF, Zachman,[6] Integrated Architecture Framework (IAF) [474], or Generalized Enterprise Reference Architecture and Methodology (GERAM),[7] as well as specific tools that may support one or more modeling language(s) but are not a modeling technique by itself, such as ARIS Cloud,[8] Bizzdesign,[9] OpenModeling,[10] or the Open Model Initiative (OMI),[11] are left out. The C4model[12] is also not included in the analysis as it does not prescribe a notation, but relies on existing modeling techniques such as ArchiMate and UML. Examples of the provided modeling techniques can be found in Appendix A.

In Table 4.1 the scoring of the enterprise modeling techniques with respect to the defined criteria (Section 4.1.2) is presented. For the scoring with respect to the criteria an ordinal scale is applied: '-' if it is clear that the criterion is not met, '0' if the criterion is somewhat met, and '+' if the criterion is reasonably or fully

[3] https://prince2.wiki/
[4] https://www.opengroup.org/togaf
[5] See, e.g., https://en.wikipedia.org/wiki/Agile_software_development#Agile_software_development_methods.
[6] https://www.zachman.com/about-the-zachman-framework
[7] Currently part of ISO19439: https://www.iso.org/standard/33833.html.
[8] https://ariscloud.com/
[9] https://bizzdesign.com/
[10] https://open-modeling.sourceforge.io/
[11] https://www.omilab.org/
[12] https://c4model.com/

met. The scoring is based on examining existing and easily accessible literature and on practical experience of the researcher with the modeling techniques.

4EM

For Enterprise Modeling (4EM) is created as a high mature, openly available method that focuses on techno-social systems, includes all perspectives of enterprise modeling and supports participative modeling [394, 279]. It mostly comprises a systematic way of working (WoW) and a notation with a well-defined metamodel. Despite its intent to cover all perspectives, 4EM does not seem to have a product and service perspective, and the information or data perspective seems limited to terms only. The number of cases that can be found is limited, and only a few tools (partially) support 4EM.

ArchiMate

ArchiMate[13] is a visual modeling language for describing, analyzing and communicating Enterprise Architecture [277, 449]. It comes with a framework or reference structures that classifies all concepts of the ArchiMate language. Since 2008, ArchiMate is a standard of The Open Group.[14] Because it is an open standard, there are several communities of active practitioners and researchers around ArchiMate.[15]

ArchiMate supports TOGAF by providing elements for each ADM phase, from strategic capabilities to business processes, objects, services, products and actors, to application components and technology nodes, including motivation (why), and migration planning (how to get there), including relationships between all these elements. ArchiMate has a formally defined metamodel and a standardized XML-based exchange file format that is supported by several tools. Its semantics are defined in an informal way.

It has been suggested that ArchiMate has some drawbacks: its metamodel is multi-interpretable, contains irrelevant concepts while other (important) concepts may be missing [135, 20]. This might be the result of the fact that the theoretical background (WoT) behind ArchiMate seems to be missing. ArchiMate does not contain a structured way to capture business rules.

Architecture of Integrated Information Systems

The Architecture of Integrated Information Systems (ARIS)[16] framework provides an approach to support the entire system development life cycle of enterprises, from conceptualization to IT implementation [400]. It uses five perspectives or views: function, organization (including human-beings), data, product and services (also known as output), and processes (also known as control). It relies

[13] https://www.opengroup.org/archimate-forum
[14] https://www.opengroup.org/
[15] E.g., https://community.opengroup.org/archimate-user-community/home.
[16] https://en.wikipedia.org/wiki/Architecture_of_Integrated_Information_Systems

(partly) on existing modeling languages such as Entity-Relationships, EPC and BPMN.

ARIS has a community of active users, and is supported by several tools. It is unclear what theoretical foundations (WoT) are behind ARIS. ARIS does not support the modeling of business rules.

Business Process Modeling and Notation

The Business Process Modeling and Notation (BPMN)[17] is a standard by the Object Management Group (OMG) that defines the graphical representation for specifying business processes. It has been suggested that it is closely related to UML Activity Diagrams [484]. The BPMN specification also provides a mapping between notation and the underlying constructs of Business Process Execution Language (BPEL) [484].

There are many tools that support the creation of BPMN models. Most BPMN communities seem to be organized around specific tooling. While BPMN only focuses on modeling business processes, additional standards have been adopted by the OMG to support case management modeling (Case Management Modeling and Notation[18]) and decision modeling (Decision Modeling and Notation[19]). Several studies however show that these standards are not (yet) integrated [201, 354, 108]. While BPMN is mostly a notation, there is a metamodel but its semantics seem to be poorly defined [370, 126, 493, 256, 86]. It does not seem to build upon a theoretical background (WoT).

CogNIAM

Cognition enhanced Natural language Information Analysis Method (CogNIAM),[20] a successor of NIAM, is a conceptual fact-based modeling method to structure and classify knowledge. It that aims to integrate different dimensions of knowledge, i.e., data, rules, processes and semantics, and uses BPMN, Decision Modeling and Notation (DMN) and Semantics of Business Vocabulary and Business Rules (SBVR) to do so. See the specific sections on BPMN and SBVR for details. CogNIAM will be scored as the combination of both modeling techniques.

Design and Engineering Methodology for Organizations

Having strong methodological and theoretical roots, including but not limited to Language/Action Perspective [482], Speech Act Theory [14, 406] and Theory of Communicative Action [191], the Design and Engineering Methodology for Organizations (DEMO) sets communication as the primal notion for the design of enterprises and its supporting software systems [473]. In its current state, DEMO mainly focuses on creating a so-called ontological model of an enterprise[21], that

[17] https://www.omg.org/spec/BPMN/

[18] https://www.omg.org/spec/CMMN/

[19] https://www.omg.org/spec/DMN

[20] https://en.wikipedia.org/wiki/Cognition_enhanced_Natural_language_Information_Analysis_Method

[21] More about Enterprise Ontology and its theoretical background can be found in Section 4.2.

defines the products and services that the enterprise delivers through actors, including the underlying processes, information and business rules, independent of its technological implementation [124]. This ontological model is called the essence of an enterprise, following the earlier provided definition of essential (see Section 3.1.1). DEMO's Way of Working is called Organizational Essence Revealing (OER).

There is an increasing uptake of applying DEMO in practice, illustrated by the active (but small) community and certification institute[22] as well as the reported cases regarding the use of DEMO (see, e.g., [364, 16, 107] and below) and integration with other mainstream enterprise modeling approaches such as ArchiMate [135, 103] and BPMN [469, 461, 353, 71, 321, 183]. Reported benefits of DEMO include: *a)* ensuring completeness and helping to find omissions and ambiguous situations [324, 124], *b)* providing a solid, consistent and integrated set of (aspect) models [124], *c)* creating a shared understanding [324, 106], *d)* providing a stable base to discuss required flexibility [458], *e)* supporting simulation experiments [415, 104] *f)* offering a good Return On Modeling Effort (ROME) [324, 339, 340, 124], and *g)* offering a good basis to define requirements and design or generate software systems [138, 324, 171, 220, 465, 101, 106, 124].

A reported downside of DEMO is that due to the level of abstraction, i.e., they are technology-independent, DEMO models are hard to understand and far from actual implementation and thus should be complemented with other techniques or models in order to communicate them to stakeholders and to provide the necessary implementation details [220, 398]. There are however also case studies where DEMO has successfully been used in communication with stakeholders ranging from people at the workplace [252] to software developers and business analysts [106] and even higher management [346]. The case studies above have shown DEMO models can be used for software design and implementation. Another concern is that there is a lack of (free) modeling tools for DEMO [415]. Currently available tools for modeling DEMO models are OpenModeling,[23] Simplified Modeling Platform,[24] and an extension (MDG) for Sparx EA.[25]

Although there are some initiatives to enhance DEMO and/or its notation [359, 358, 349, 350, 15], the current version of DEMO Specification Language (DEMO-SL) and its metamodel are documented in [122, 325].

Event-driven Process Chain

Event-driven Process Chain (EPC) is a type of flow chart for business process modeling and part of ARIS. Main elements of EPC include: event, function (process (step) or task and decision), process owner, information and connectors between these elements. Its metamodel is only informally described and (thus) lacks proper semantics [460]. Suggestions to improve EPC have been made [314, 450], but it is unclear whether these suggestions have been adopted. The EPC com-

[22] https://ee-institute.org
[23] https://open-modeling.sourceforge.io/
[24] https://teec2.nl/products/modelling-platform/
[25] http://www.eaxpertise.nl/cmsform.aspx?webpage=DEMO4MDG

munity seems to reside within the (broader) ARIS community. It is mainly the ARIS tools that support EPC, although some others can be found as well.

Multi-Perspective Enterprise Modeling

Multi-Perspective Enterprise Modeling (MEMO) is an academic initiative that incorporates several perspectives, ranging from processes to organizational structure to information models [161]. Additionally, a metamodel, modeling language and tools have been developed [162, 164, 163, 166, 49, 48]. Although MEMO provides a well documented metamodel, it seems to be missing a modeling procedure (WoW) and theoretical foundation (WoT). As MEMO is merely an academic initiative that is not (widely) being used in practice, it lacks a community and cases cannot easily be found.

Semantics of Business Vocabulary and Business Rules

SBVR[26] is a standard of the OMG, intended to be the basis for a formal natural-language based description of enterprises. It builds upon the fact-oriented approach that is designed to promote correctness, clarity and adaptability and enhances semantic stability [195]. SBVR uses controlled natural language to express business vocabulary, i.e., terms, and business rules in a declarative way, i.e., *what* it should be or do, not *how*. Rulespeak[27] provides a set of guidelines for expressing business rules that is fully compliant with SBVR. The SBVR community seems to be small, and not a lot of example cases can be found.

System Modeling Language

System Modeling Language (SysML)[28] is a general-purpose modeling language for systems engineering that supports specifying, analyzing, designing and validating systems. It was originally developed as an open-source project and later adopted by the OMG. It is defined as an extension of a subset of UML and includes nine types of diagrams of which some are taken from UML.

Unified Modeling Language

UML[29] is a standardized modeling language intended for the specification and visualization of software. It consists of best practices and a (graphical) notation and is maintained by the Object Management Group. UML provides two categories of diagrams: structure and behavior. Structure diagrams represent the static aspects of a system, such as components and classes. Behavior diagrams represent the dynamic or functional aspects of a system, such as activities, use cases and communication.

[26] https://www.omg.org/spec/SBVR/
[27] https://www.rulespeak.com/en/
[28] http://www.omgsysml.org/ and https://sysml.org/
[29] http://www.uml.org/

	Way of Modeling	Way of Thinking	Way of Working	Way of Supporting	Products & Services	Actors	Processes	Rules	Information or data	Documented metamodel	Formal semantics	Number of example cases	Community size
4EM	+	-	+	0	-	+	+	+	+	0	0	0	0
ArchiMate	+	-	0	+	+	+	+	-	+	+	0	+	+
ARIS	+	-	-	+	+	0	+	-	+	+	0	+	+
BPMN	+	-	-	+	0	0	+	0	0	+	0	+	+
CogNIAM	+	0	0	+	-	-	+	+	+	+	+	0	0
DEMO	+	+	+	0	+	+	+	+	+	+	+	0	0
EPC	+	-	-	+	-	-	+	0	0	+	0	0	+
MEMO	+	-	0	0	-	+	+	0	+	0	0	0	-
SBVR	+	+	0	0	-	0	-	+	+	+	+	0	0
SysML	+	-	-	0	-	0	0	-	0	+	0	0	0
UML	+	-	-	+	0	0	0	0	+	+	0	+	+

Table 4.1: Overview of enterprise modeling techniques scored to the identified criteria (see Section 4.1.2)

UML has a big user group and many tools support this standard. And although UML is being used for enterprise modeling, it is also noted that *a)* it has too many constructs which makes it complex and hard to learn or use [34, 160], *b)* it lacks proper concepts for enterprise modeling [511], and *c)* its semantics are (very) poorly defined [242, 137, 258, 77, 190, 159, 192, 275, 62].

4.1.4 Choice for This Research

As can be seen in Table 4.1, CogNIAM, DEMO, and SBVR seem to be the only modeling techniques that are (somehow) based on a Way of Thinking and with a formal semantics. On the other hand, 4EM, ArchiMate, and DEMO seem to be the only modeling techniques with a proper WoW and covering all or most modeling perspectives that are important for this research. It seems logical to choose DEMO as input for the EMDSD approach as applied in this research, as it seems to score best with respect to the chosen criteria.

However, the lack of tools to create DEMO models is a concern. An alternative is to combine, possibly multiple to cover all perspectives, modeling techniques for which there are proper tools with the WoT of DEMO or SBVR. As said before, creating such a technique can be a research on its own and therefore DEMO is chosen as it has a formal semantics and covers all perspectives, and the downside

is accepted that for this research the models have to be created and converted into a format that a computer program can understand manually.

The other issue that DEMO or ontological models are abstracted a lot from (enterprise) implementation can be regarded as both an advantage and disadvantage: only by abstracting away from implementation, it can be used to truly redesign the implementation, while in implementation-dependent models optimization is the best achievable result. It is however also a concern with respect to the EMDSD approach as adopted in this research. In order to bridge the gap from enterprise ontology to implementation, the Enterprise Implementation Framework (EIF) is created with the Organization Implementation Variable (OIV) as main concept to describe an enterprise's implementation that is linked to the ontological model of the enterprise. The EIF is thus fully compliant with the GSDP and has been validated during the exploratory case studies in Part II. Both Enterprise Ontology and Enterprise Implementation are detailed in the next sections.

4.2 Enterprise Ontology

Building upon the definition of ontology in general (see Section 3.1.1), the goal of enterprise ontology is to share knowledge about an enterprise within and between enterprises [456]. As enterprise knowledge is local by nature, it needs to be specific for a certain enterprise [284]. Bertolazzi et al. have composed a core enterprise ontology describing generic enterprise concepts [35], but they do not offer a way to capture the ontology for a specific enterprise.

Building upon the Language/Action Perspective [482], Fox proposes actors, their roles and communication links between them as key concepts in enterprise ontology [157]. In his EE-theories for Enterprise Engineering and Enterprise Ontology, Dietz builds upon these concepts by seeing an enterprise as a network of actors that enter into and comply with commitments [124]. Such commitments are raised by actors in acts, the atomic units of action, and follow a generic pattern called the Complete Transaction Pattern (CTP). The general working principle is that actors constantly check whether there are acts they have to deal with or respond to, the so-called *actor cycle*. By abstracting actors to actor roles and commitments regarding a specific product to transaction kinds, the model becomes independent of the people involved in the operation.

According to the EE-theories, an enterprise ontology describes both the dynamics and statics of an enterprise and concerns the highest level white-box model of the construction and operation of the organization of an enterprise [124]. Since it only depends on an enterprise's products and services, it is fully independent of the way in which it is realized and implemented [119]. These models are therefore considered more stable than implementation dependent models [123]. An enterprise ontology is expressed in a set of aspect models that is claimed to be comprehensive, coherent, consistent, and concise [124, p. 14].[30] DEMO is a

[30]While it is not possible to prove these claims, several case studies have confirmed, or at least not been able to disprove, these properties, see, e.g., [138, 324, 339, 171, 220, 364, 340, 465, 16, 101, 107, 415, 458, 104, 124].

leading method in the field of Enterprise Engineering, currently mainly covering Enterprise Ontology; the ontological models of an enterprise are sometimes referred to as DEMO models.

According to the EE-theories, three layers of organizations can be discerned in enterprises [124]: *a)* original for devising things, deciding, manufacturing, and transporting things, *b)* informational for remembering, computing, deriving and sharing facts, and *c)* documental for saving, providing, transforming and deleting documents, data and files. The ontological model of the original layer of an enterprise is also called its essence.

In this chapter the notions of CTP (Section 4.2.1), actor cycle (Section 4.2.2 and organizational layering (Section 4.2.3) are introduced. Moreover, the ontological aspect models (Section 4.2.4) and the DEMO metamodel (Section 4.2.5), needed to define the model transformation for the EMDSD approach (see Section 3.3.2) as adopted in this research, are detailed.

4.2.1 Complete Transaction Pattern

According to the EE-theories, each commitment is raised in a coordination act (C-act), e.g., 'Martin requests Erik order #125[31] is completed', and results in a corresponding coordination fact (C-fact). Following Habermas' Theory of Communicative Action [191], every C-act has a performer (in the example: Martin), an addressee (in the example: Erik), an intention from the CTP (in the example: request), and is about some product or production fact (P-fact) (in the example: 'order #125 is completed'; see Figure 4.2). An act can be performed explicitly, i.e., by verbal or non-verbal communication, or implicitly, i.e., tacitly as there is no explicit act but its existence can be deduced from the presence (or absence) of other acts [472]. C-acts (and its corresponding C-facts) about the same product are called transactions and all follow the CTP. The CTP is considered to be the universal pattern in all organizations [124]. It consists of 18 C-act kinds (intentions) and 1 production act (P-act) – in which the product is being produced – and deals with the basic flow – request, promise, execute, declare and accept – as well as discussion states – decline, reject – and cancellations, called revocations (see Figure 4.3).

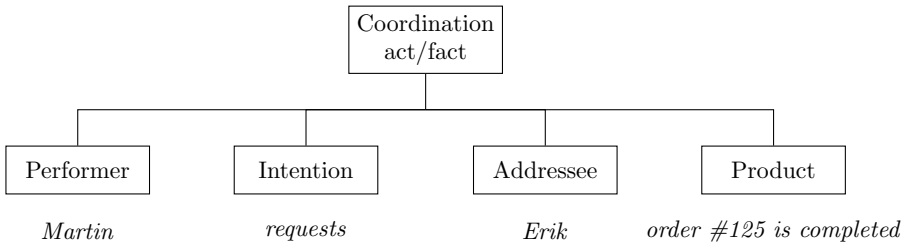


Figure 4.2: Structure and example of a C-act and C-fact, adapted from [124]

[31] The order can have many attributes, like exactly what is being ordered and when the result is expected. For simplicity, these properties are left out.

C-acts (and its corresponding C-facts) are considered the atomic building blocks of organizational processes, while transactions are considered the molecular building block of organizations. Every transaction instance is of a particular transaction kind, e.g., 'order completing'. A transaction kind concerns one specific product kind, e.g., '[order] is completed', has one specific actor role, e.g., 'order completer' as its executor role, and can have multiple actor roles as its initiator role. The performer of the request of some transaction is the initiator of that transaction, while the addressee of the request is the executor of that particular transaction.

The CTP tells that after a request, the executor (in the example: Erik) can choose between performing a promise or decline. On the level of the C-act, the roles of performer and addressee will be reversed: Erik is the performer and Martin the addressee, e.g., 'Erik promises Martin order #125 is completed'. Instead of promising, a decline is possible which brings the transaction into a discussion state in which the initiator and executor of the transaction have to sit together and discuss next steps. One possibility is that the initiator convinces the executor to agree to the request, after which the request will be renewed so that the executor can promise it. Another possibility is that together they negotiate the terms for a (slightly) changed request of the initiator that the executor can then promise. A last, but often not desired, possibility is that the discussion end with a disagreement, not resulting in a renewed request. A similar reasoning holds for the discussion state 'rejected', where the roles are reversed (again).

The CTP also defines revocation patterns that allow actors to revoke (withdraw or cancel) a C-act. The revocation pattern in itself is generic but made specific for the four 'basic' C-acts (see Figure 4.3) – from the discussion states it is always possible to get back to the main flow, so no revocations are needed for them. It is a separate flow as it does not immediately influence the main flow. However, it is best practice stopping the work in the main flow once a revocation flow has been started. The idea is that either the initiator or the executor of the transaction can start a revocation, after which the other actor can decide to allow the revocation or to grant it. The start of the revocation is also a discussion state, as usually both parties have to discuss the terms with which a revocation can be allowed. It will mostly depend on the specific situation at hand, and whether the executor has already started producing the P-fact whether a revocation will be allowed or refused. For example, when Erik has already produced the order, it will be unlikely that he allows Martin to revoke the request. Only when the revocation is allowed, the main flow returns to a previous state, as indicated in the CTP.

The CTP can be compressed by only showing the involved actor roles and the transaction kind it is about, abstracting from but still implying the underlying CTP. Actor roles are denoted by a box while transaction kinds are denoted by a diamond-in-a-disk (see Figure 4.4). A transaction kind and its executing actor role together is called transactor role. These symbols are used in the Cooperation Model (CM) as will be explained in Section 4.2.4.

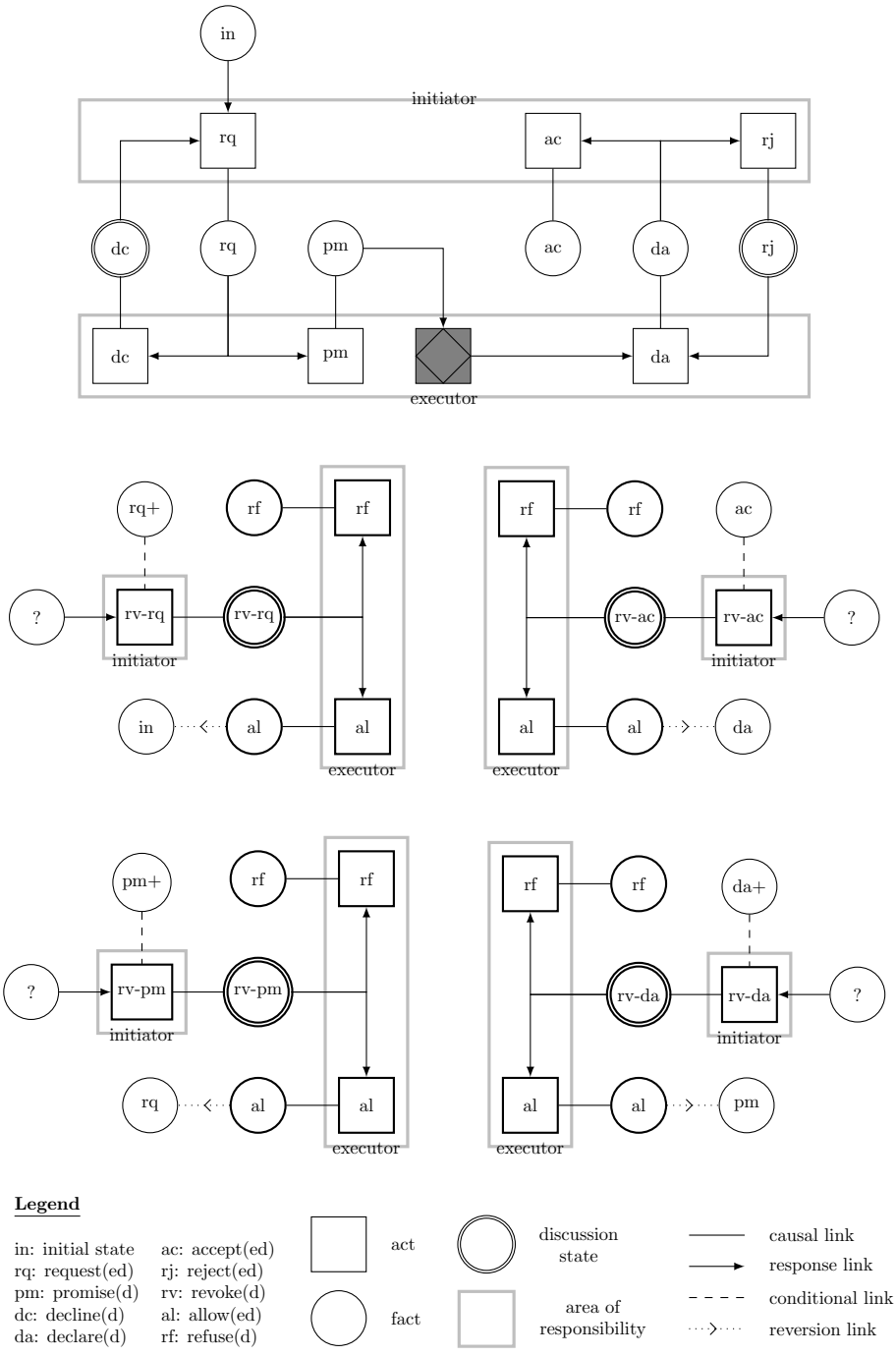


Figure 4.3: Complete transaction pattern comprising the basic flow and four revocation flows, adapted from [124]

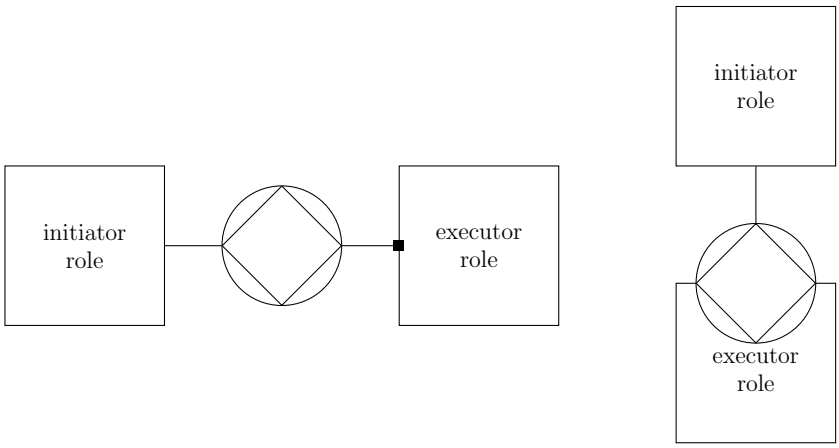


Figure 4.4: Denotation of actor role and transaction kind. For compatibility reasons, both the DEMO-3 notation (left) and DEMO-4 notation (right) are shown. Only the latter explicitly shows the concept of transactor role, while it is implied in the older notation by the black box on the executor role.

4.2.2 Actor Cycle

Every actor is considered to constantly loop through its operating cycle (see Figure 4.5), at a pace that is sufficiently frequent to deal with the agenda on time [124]. The actor's agenda consists of C-facts (see Section 4.2.1), the total set of acts for the actor to deal with.

The cycle starts with an actor selecting an agendum (a single item in the agenda) to be settled – at this stage it is irrelevant how such an agendum is chosen, typically based on internal or external defined priorities. Then, the actor fetches the applicable action rule(s) (from the Action Model (AM), see Section 4.2.4). Next, the actor assesses the situation following Habermas' three validity claims [191]:

- in the *claim to rightness* the actor checks whether both parties have the authority to be the performer and addressee of the C-act;
- in the *claim to sincerity* the actor verifies the trust and honesty of the other party;
- in the *claim to truth* the actor checks for possible violation of rules.

In order to verify the claims, the actor usually needs to fetch information saved earlier or by someone else. After the assessment, the actor decides how to respond to the selected C-fact. Typically, when all claims have been successfully verified, the actor will continue with the basic flow. Otherwise, the actor will move the transaction into a discussion state. Either way, the actor performs the act(s) that follow from the decision.

With reference to Figure 4.5, let's look again at the example: Suppose Martin has performed the C-act as stated earlier, viz., 'Martin requests Erik order #125

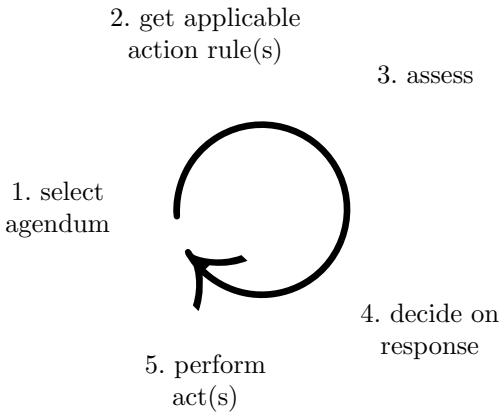


Figure 4.5: Operating cycle of actors, adapted from [124]

is completed'. Now, there is a C-fact on Eriks agenda. After having had his morning coffee, Erik notices this C-fact is the only agendum for the day. Thus, he chooses to act upon this agendum (1). He fetches the applicable action rule that guides him in his decisions (2). The action rule (see Section 4.2.4 for more details) tells Erik to assess the situation (3):

a) he checks whether he is authorized to deal with the request – as he is handling it, this is assumed to be true;

b) Erik checks whether Martin was allowed to perform the request – as Martin is a potential customer and does not seem to be a crook, let's assume this is true as well;

c) then he verifies the trust and honesty of Martin – let's again assume this to be true as well;

d) lastly, Erik checks whether the enterprise allows producing the P-fact, i.e., to complete the order. Examples in which the order cannot be completed are when the product is not there to complete the order, when Martin expects the order to be completed within 12 hours but the current planning only allows for completion after 48 hours, or when Martin does not have the right certification to receive the completed order.

Having assessed the situation, Erik decides (4) to promise the order completion and, as a result, performs (5) the C-act 'Erik promises Martin order #125 is completed'.

As actors are autonomous in deciding how to act, action rules must be understood as guidelines. The responsibility given to an actor in fact allows the actor to, responsibly, make a decision that is not in line with the action rule(s). In such

cases, the actor is expected to be able to explain the reason of deviating from the action rule.

4.2.3 Organizational Layering

In an enterprise, three different types of (trans)actors can be distinguished [124]: original, informational, and documental. This view is based on the semiotic ladder that distinguishes between commitments in the social world (orginal) from semantics or content (informational) and syntax or form (documental) [286].

Original (trans)actors create new facts by, e.g., devising, deciding, judging, manufacturing, transporting or observing things.

Informational (trans)actors support the original (trans)actors by remembering, deriving, computing and sharing facts.

Documental (trans)actors support the informational (trans)actors by saving, providing, transforming and deleting documents, data and files.

These layers should be seen as distinct (organizational) layers: O- (O for Original), I- (I for Informational) and D-organization (D for Documental). Put together, these organizations are the enterprise. For all three layers, an ontological model can be created. The ontological model of the O-organization is called the enterprise's essence. There are procedures that describe how to devise (or generate) parts of the ontological model for the I-organization based on the ontological model of the O-organization, and similarly for creating parts of the ontological model for the D-organization based on the ontological model of the I-organization [102, 101, 124].[32] In practice, however, these models are barely created as software can easily be created from the ontological model of the O-organization [117].

While informational and documental acts and actors can easily be taken over by IT artifacts, such as software systems, only human actors can be held responsible[33] for acts. Original acts should only be performed by human beings and it should always be clear which human actor is responsible for the IT artifacts that perform informational and documental acts.

4.2.4 Ontological Aspect Models

The ontological model of an organization consists of an integrated whole of four aspect models[34] (see Figure 4.6):

[32] The generation is only partly as there may be non-derivable I- and D-transaction kinds.

[33] Most organizations make a distinction between responsibility and accountability. In this research they are treated as synonyms.

[34] It is doubtful whether 'aspect model' is the right term. In general, a model is a simplified representation of a (software) system to study aspects of a system (see Section 3.1). The system to be studied is an enterprise, the model to do so is its ontological model. As the ontological aspects models as defined here form an integrated whole, it could be argued that instead of aspect models, they should be called model aspects or (model) viewpoints.

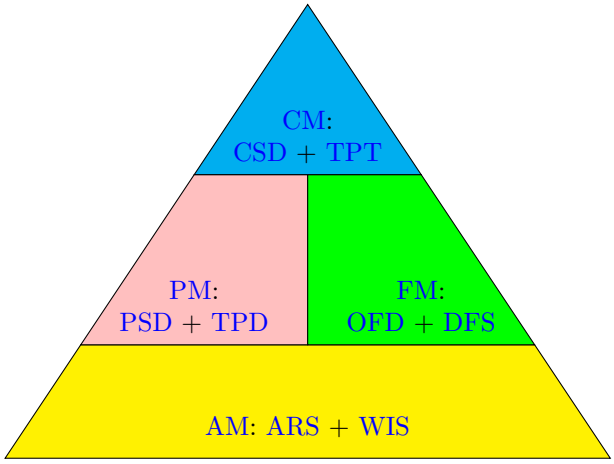


Figure 4.6: Ontological aspect models, adapted from [124]

Cooperation Model (CM) models the *cooperation* of the enterprise; it consists of transaction kinds, associated (initiating and executing) actor roles, fact banks, the access links between actor roles and fact banks, and wait links between transaction kinds and actor roles. The CM is expressed in one or more Coordination Structure Diagrams (CSDs) and a Transactor Product Table (TPT);

Process Model (PM) models the *processes* that take place as the effect of acts by actors, by detailing the coordination between actor roles. It specifies the state and transition space of the *coordination world* by making explicit the causal and wait links between C-(f)acts from the CTP. The Process Model (PM) is expressed in one or more Process Structure Diagrams (PSDs) and one or more Transaction Pattern Diagrams (TPDs);

Fact Model (FM) is the semantic model of *products* of the enterprise – specifying the state and transition space of its *production world* in terms of fact types (entity types with their related product kinds, property types, attribute types and value types), existence laws and occurrence laws. The Fact Model (FM) is expressed in an Object Fact Diagram (OFD) and zero or more Derived Fact Specifications (DFSs);

Action Model (AM) is a model of the *operation* of the enterprise, guiding actors in performing P-acts (through Work Instruction Specifications (WISs)) and C-acts (through Action Rule Specifications (ARSs)). It specifies for every C-fact kind with which the enterprise has to deal – called agendum kind – one or more ARSs. Each ARS contains three parts that support the actor cycle:

- an *event part* consisting of a when-clause stating the agendum kind to respond to, a while-clause if additional wait conditions (impediments)

apply, and a with-clause that specifies the fact kinds that are needed to assess the action rule;
- an *assess part* to check for certain conditions, following the 3 claims from Habermas (see Section 4.2.2); and
- a *response part* that states how the actor should respond, possibly defining an alternative path.

4.2.5 DEMO Metamodel

DEMO-SL is the formal language in which the ontological models as defined in Section 4.2.4 are expressed. In Figure 4.7 the relevant parts of the metamodel for this research are depicted. The general step kinds are defined by the CTP (see Section 4.2.1) and thus are not defined in any of the aspect models. Product kinds are defined in both the CM and FM. A derived fact type can be expressed as a calculation, which can be an aggregation, specialization, or generalization. Both (elementary) actor roles and fact types can be in or out of focus – composite (trans)actor roles and multiple transaction kinds are by default out of focus, while the other concepts are by default in focus. Out (of) focus means the concept is shown in the higher-level models, such as CM, PM, and FM, but no (detailed) action rules are created.

While there is an Extensible Markup Language (XML)-based exchange model for DEMO available [325], for this research a more compact JavaScript Object Notation (JSON) format to represent the input models for automated model conversions is created. This version leaves only contains item from the metamodel does not contain parts that are for representation only.

4.3 Enterprise Implementation[35]

In order for an enterprise to become and stay operational, it needs to be implemented with appropriate technology in a future-proof way. This implies that the construction model of an enterprise is described at such a level of detail that it can be put into operation. In doing so, it should be possible to consider multiple alternatives and choose the better (or optimal) one with respect to some business goals or objectives. It can even turn out that different alternatives can co-exist, or that one implementation should be able to evolve into another implementation relatively easily. While it can be very costly to keep every option open, it is also not necessary; a pizzeria will not suddenly become an insurance company or the other way around. Instead, the need for adaptability in implementation should be identified carefully. It is thus needed to better understand how the implementation model looks like and how it can be used to design and compare alternatives as well as informedly decide on the variability in implementation that can exist over time or over different locations.

[35]Parts of this section are originally published as M. R. Krouwel, M. Op 't Land, and T. Offerman, *Formalizing Organization Implementation* [265]. A detailed mapping from the original paper to this thesis can be found on p. xxx ff.

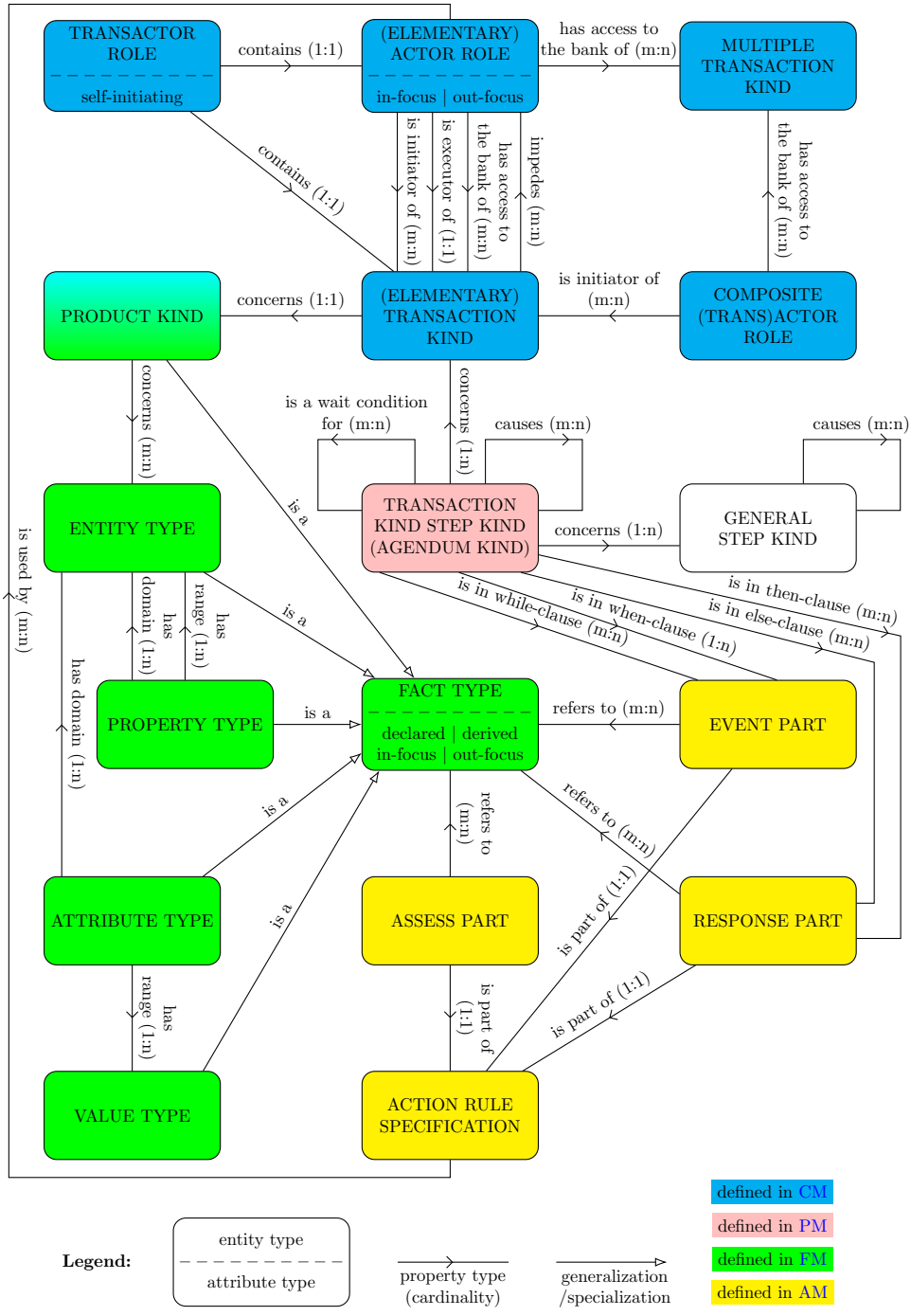


Figure 4.7: DEMO metamodel, adapted from [122, 325]

As outlined in the GSDP (see Section 3.2), the construction model of any system, including enterprises, comprises both the ontological model and the implementation model, the lowest level construction model of the OS, fully detailing the specifications of a possible implementation that can subsequently be implemented using appropriate technology [120]. When a product or service is decided upon, and the collaboration network needed for its delivery has been revealed in the ontological (enterprise) model, still many degrees of freedom exist before an enterprise can become operational [124, p. 46]. While architecture constrains the design freedom (see Section 3.2.4), still many technological alternatives can satisfy both the functional (user) and constructional requirements. The remaining choices can and should be captured as design decisions. Each decision can be expressed as value for a certain (organization implementation) variable, called OIV. Together, these implementation design choices constitute the lowest level and most detailed model of the construction of an enterprise.

This section introduces the Enterprise Implementation Framework (EIF) that details the layers that can be discerned in implementation (Section 4.3.1), shows some examples (Section 4.3.2), highlights the benefits of using the approach of expressing an implementation in terms of variables (Section 4.3.3) and ends with implications for software development (Section 4.3.4).

4.3.1 Implementation Layers

Implementation spans different layers:

Organization: the (non-ontological) structure of the organization such as functionary types, organizational units, work locations, and the types of means that support the actors in doing their work, as well as the relations between them and with the ontological elements (mainly agendum kind, see Figure 4.7);

Means: all technological means, including human beings and IT artifacts – also known as silicon and carbon servers [454] – as well as tools, equipment, buildings, vehicles, office supplies, etc., that are needed to operate;

Installation: the (temporary or more durable) assignment of specific means from the means layer to (abstract) elements from the organization layer; and

Operation: the assignment of specific agenda (see Section 4.2.2) to specific means.

In fact, with reference to the model kinds as described in Section 3.1.1, the ontological model can be considered to be the conceptual model of the enterprise, the organization implementation layer can be considered the logical model of an enterprise, and the other implementation layers can be considered the physical model. The latter two combined constitute the implementation model. This layering therefore helps in classifying the different types of change as described in need 2.

An OIV can be elementary or cross-reference. An elementary OIV does not depend on the existence of some other OIV or element in the enterprise ontology, e.g., functionary type. A cross-reference OIV does depend on the existence of some other OIV or element in the enterprise ontology, e.g., authorization as the assignment of a functionary type to an actor role.

By expressing the implementation as a set of values for a (chosen) set of OIVs, a new implementation alternative can be created simply by changing the value of a single variable. These alternatives can then be evaluated against the requirements, e.g., by means of simulation. In practice, the implementation of organization, and therefore the value(s) of OIVs, will change far more often than the products and services they help deliver.

4.3.2 Examples

Extensive literature review on the topic of Enterprise Implementation, e.g., [96, 397, 455, 84, 119, 345, 414, 459, 470, 215, 344, 391, 402], enterprise or process flexibility [287, 371, 404, 194], as well as assessments on both academic [334, 121, 341] and real-world use cases [317, 458, 104] on the presence of OIVs, has resulted in a list of ∼30 variables [336].

Examples include (see Figure 4.8):

- deciding on *work locations* and *organizational units*, e.g., which branches and departments exist;
- deciding upon *functionary types* and the *authorization* that describes which functionary type fulfills which actor role(s) or deals with which agendum kind(s) – e.g., the functionary type 'cook' that fulfills both the actor roles 'baker' and 'stock controller'; or the functionary type 'deliverer' who is authorized for the acts 'promise' and 'declare' of the transaction kind 'transport' and also for the act of 'accept' for the 'customer payment' transaction kind;
- deciding on which C-acts are performed explicitly and which implicitly (see Section 4.2.1); and
- deciding on *order of working* and *logical units of work* – e.g., should delivery only be done after receiving payment (as common in retail) or is it (also) possible to pay afterwards (more common in B2B), and should receiving of payment and delivery be seen as a single unit of work that should be carried out by the same person without interruptions.

4.3.3 Benefits

By expressing an enterprise implementation model as a set of values for a (chosen) set of variables, it can be used to

a) decide informedly upon enterprise changes,

b) enable traceability in governing transformations,

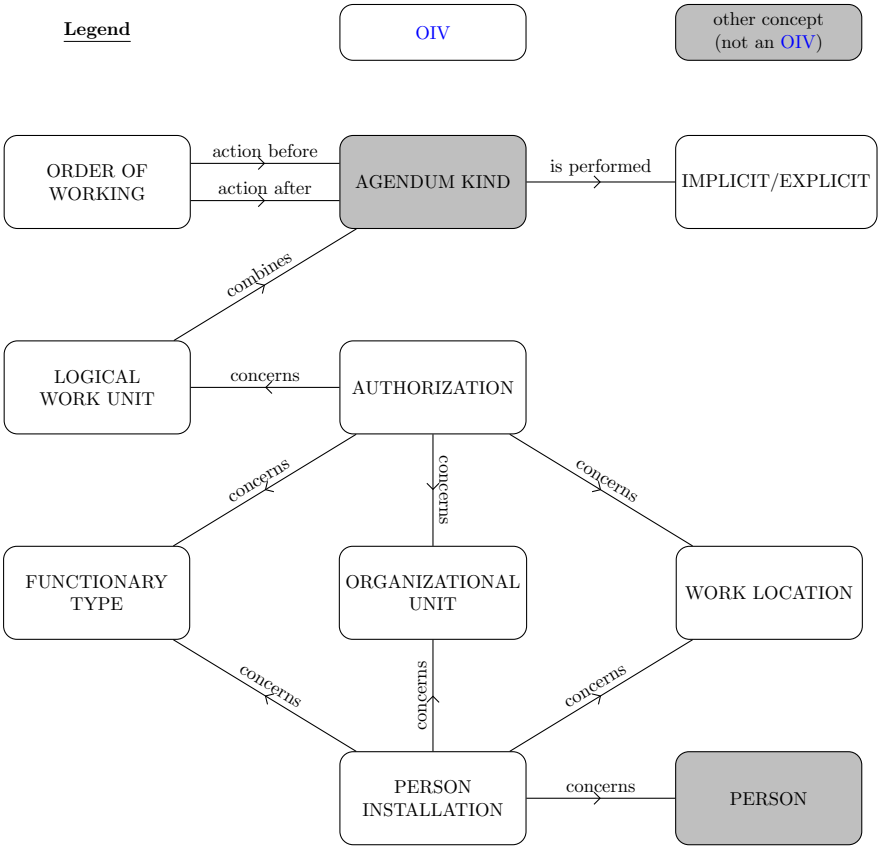


Figure 4.8: Example OIVs and their relations

c) assess to what extent software solutions support enterprise implementation variability,

d) design software systems that inherently supports enterprise variability, and

e) design enterprises that support variability.

4.3.4 Implications for Software Development

In order to achieve enterprise agility, it is a priority that frequently occurring changes are not on its critical path (see Section 1.1 and need 2 specifically). Ideally, it is possible to make such changes with no or only little impact in the supporting software systems. This may show the need for an implementation to be easily adaptable in some variables, while for other variables it is not necessary to change its value easily. E.g., departments may be rearranged yearly, while work locations may be more stable. Analyzing alternatives may create the (architectural) requirement that the value for a certain variable is easily changeable,

imposing a design constraint that can heavily impact the design of supporting software. As a minimum requirement, it should be clear which software components are affected by a change in the (enterprise) implementation, and, preferably, this number of components is limited.

On the other hand, organizationally variability can be quite high, making it difficult to gain overview and impossible to support all possibilities. With the simplistic assumption that each of the (14) elementary OIVs can have 3 different and each of the values is able to change independent of the values of the other OIVs, the total number of possible organizational implementation alternatives is already $3^{14} \approx 4.8 * 10^6$, per transaction kind! This is a problem for both human understanding and automation; it is thus necessary to make conscious choices on what changes should be supported by the supporting software systems and to what extent.

Organization Implementation Variables are a way to make explicit the organization design decisions that need to be supported by the software systems. The Enterprise Implementation Framework also provides the possibility to make an explicit (architectural) requirement that the value for an OIV should be (easily) changeable in the software. In a way, it is similar to Normalized Systems Theory, but it defines expected types of change on the enterprise level and not in terms of software constructs.

4.4 Conclusions

In this chapter the notion of Enterprise Modeling in general as a means to understand what an enterprise does and as input for the software development process has been explored. Different perspectives can be chosen when modeling an enterprise, and combining multiple perspectives is key in order to achieve agility and to improve the quality of the enterprise models. Based on the general need in this research for qualitative models, the specific needs for the EMDSD approach as adopted in this research, and the need for different perspectives, several criteria have been formulated to choose a modeling technique that supports the purpose of this research. Several modeling techniques are outlined and, while it is possible to combine different techniques to fully cover all important criteria, DEMO models have been chosen as input for the EMDSD approach as adopted in this research, that now becomes an Enterprise Ontology-Driven approach towards Software Development.

DEMO is well-grounded in theory and provides the ontological model of an enterprise, covering all perspectives that are considered essential for this research. This technique is fully compliant with the GSDP (see Section 3.2), and provides enterprise models with a formal semantics as is required for MDSD. The downsides of using DEMO is the lack of proper tooling – which is not an insurmountable hurdle – and the big gap towards (software) implementation. In order to deal with the latter issue, the Enterprise Implementation Framework has been developed to capture organizational implementation design decisions in terms of Organization Implementation Variables. Together, DEMO and the EIF cover the different

Model Kind	Layer	Change type	Model
Ontology: descriptive, technology-independent	Conceptual	Adding or deleting a product or service	DEMO (essence)
Implementation: prescriptive, technology-dependent	Logical	Adding or deleting a department, functionary type, means type or process step	EIF: Organization layer (organizational implementation)
	Physical	Adding or deleting a specific technological resource	EIF: Means, Installation, and Operation layers (technical implementation)

Table 4.2: Overview of the model kinds, layers, types of enterprise change and used models in this research

model kinds as discussed in Section 3.1.1 as well as the typical enterprise changes as formulated in need 2 (see Table 4.2).

From theory, an initial answer to RQ 2 would be DEMO-models extended with explicit organization design decisions expressed as a set of values for some OIVs. This initial answer should be validated in practice as will be done in Part II. Before diving into the practical part of this research, in Chapter 5 the topic of (Situational) Method Engineering is detailed and an overview of the method elements found so far is provided, to guide the practical validation part of this research.

"The whole difference between construction and creation is exactly this: that a thing constructed can only be loved after it is constructed; but a thing created is loved before it exists, as the mother can love the unborn child."

Gilbert K. Chesterton, in 'Appreciations and Criticisms of the Works of Charles Dickens' (1991)

5 Towards a Method

The research challenge is to contribute to the design of a method, including supporting tools, for the development of software from the ontological model of an enterprise including explicit implementation design decisions, that can be applied repeatedly and is adaptable to specific situations. A method is a step-by-step approach to perform a system development project, based on a specific way of thinking, consisting of structured activities and deliverables, preferably supported by tools [17, 58, 199] and helps to develop software in an effective and efficient way by standardizing the process [245, 151]. Such a method can be created (and loved) before its actual construction.

In this chapter an initial answer to RQ 4 is provided by exploring the field of Method Engineering and Situational Methods in Section 5.1. Then, existing literature is reviewed for the presence of usable method elements in Section 5.2. This chapter ends with some initial conclusions and directions for the practical research in Section 5.3.

5.1 Method Engineering

Method engineering is the discipline to design, create and adapt methods for the development of software systems [272, 58]. In creating or configuring a method, often (proven) parts of existing methods [200], called *method fragments* [199], are used. Creating a method from method fragments is also known as *method assembly* and follows certain rules [59]. If such a resulting method is tailored to a specific project or particular situation, it is considered a *situational method* [200]. Especially in creating situational methods from method fragments, it is important to set the configuration and adaptation mechanisms in terms of fragment configuration rules [490].

This section will introduce the notions of method fragment (Section 5.1.1) and situational method engineering (Section 5.1.2) and ends with implications for this research (Section 5.1.3).

5.1.1 Method Fragments

A method fragment is a description of a method, or any coherent part thereof [199]. Method fragments can be created from existing methods (reverse engineering) or by construction from scratch [206]. Method fragments can be categorized on 3 axes [199, 59]:

Perspective: a fragment is either a product fragment or a process fragment:

- a *product* fragment describes a deliverable, delivered by or required within the method, such as specifications, designs, models,[1] diagrams, documents, reports, etc.;
- a *process* fragment describes an activity or task to be carried out or a stage in between them, such as high level strategies, models of the development process and detailed procedures.

Abstraction: a fragment is either conceptual or technical:

- a *conceptual* fragment is an objective description, such as model descriptions and activities;
- a *technical* fragment is a specification that has been implemented with IT. More specifically, this can be further discerned into:
  * tools, including (model and diagram) editors,
  * repositories to store models and diagrams, and
  * process managers that guide the user through the modeling and development procedures.

  A conceptual fragment can be supported by one or more technical fragments.

Granularity: a fragment can reside on one of five granularity layers, from high to low:

- *Method*, if the fragment addresses the complete method for developing the entire system, e.g., the (entire) method for Information Engineering;
- *Stage*, if the fragment addresses a segment of the life-cycle of the system, e.g., function design or testing;
- *Model*, if the fragment addresses an aspect system of an abstraction level of the system, e.g., a Data Model or a User Interface Model;
- *Diagram*, if the fragment addresses the representation of a view of a Model method fragment, e.g., a Data Flow Diagram or a State (Transition) Diagram;

[1] A description of a model is a metamodel, see Section 3.1.1.

– *Concept*, if the fragment addresses the concepts and associations of Diagram method fragments, e.g., a Process or (State) Transition.

Others mention method components [238] or chunks [382, 368] that are defined as aggregates of fragments and therefore reside on the higher levels of granularity [2, 205, 388, 207]. To prevent that method fragments are arbitrarily combined into meaningless methods, only fragments of the same category (in all three axes) may be used to assemble a (situational) method [59].

5.1.2 Situational Method Engineering

Situational Method Engineering is the subarea of Method Engineering directed towards the controlled, formal and computer-assisted construction of situational methods out of fragments [200]. It combines the benefits of control by means of standardization and reuse of existing method fragments with the benefits of flexibility by providing means to tune a method to project-specific needs into controlled flexibility [200]. The process of configuring a situational method consists of a few steps [58, 199]:

1. project environment characterization,
2. selection of method fragments,
3. assembly of fragments into a (situational) method,
4. application and validation of the resulting method.

As the application of the method can provide more insights into the characterization of the environment, or the environment can change during the application, it may turn out the method needs to be refined or adapted in iterations following the same process. Note there may be a difference between the situational method as configured and the actual method as used in an intervention [3].

5.1.3 Implications for This Research

In an initial answer to RQ 4, it is shown that methods, and situational methods specifically, heavily rely on the availability of method fragments.[2] Therefore, a first analysis of possible existing method fragments that support the creation of an enterprise model-driven software development method, before creating additional method fragments. The categorization of method fragments outlined in Section 5.1.1 will be used to define and describe the (possible) method fragments. In line the model kinds (see Section 3.1.1) and the different types of change (see Table 4.2), and in order to support the natural flow of conceptual models from high-over to more detailed (see Section 3.2) – but still independent of technology! – the abstraction category 'conceptual' is split into conceptual-abstract and conceptual-implementable, where the latter is (very) detailed, low-level, and *implementable* but not the actual IT *implementation* as a technical fragment.

[2]From hereon the term 'method fragment' is used instead of 'method element'.

5.2 Existing Method Fragments

In this section related research on creating software from enterprise ontology – as provided by DEMO – is reviewed in the search for possible fragments that could be part an enterprise model-driven software development method. The summary of findings can be found in Table 5.1.

5.2.1 DEMO to Services

Several attempts have been made to design (micro)services from ontological models. Some define a (business) service as synonymous with a DEMO transaction kind [197, 110, 486, 444], supporting the decoupling between consumer and provider as suggested by SOA [447]. Some of this research includes an example mapping to a Web Services Business Process Execution Language (WS-BPEL) [333] implementation. However, as a transaction consists of one P-act and many C-acts, involving acts performed by the initiator and executor of the transaction in an alternating way (see CTP in Section 4.2.1), this is quite coarse-grained with the risk of not being able to easily change the implementation of parts of such a service. As these ideas need further development, (micro)service creation is considered a method fragment to further explore in one of the exploratory case studies (see Chapter 6).

5.2.2 DEMO to Components

Albani and Dietz have created the Three Dimensional method for Business Components Identification (BCI-3D), a way to group business tasks and corresponding information objects into business components. This model is then used to define components from enterprise ontology [110, 8]. They however do recognize that in order to implement the identified components in software, additional implementation design decisions have to be taken. These ideas are considered a high-level method fragment.

5.2.3 DEMO to Normalized Systems

Huysmans [218] has applied the principles from Normalized Systems theory to Enterprise Ontology, as a base for automated translation from DEMO models to working software. His approach however seems to be based on a single case and lacks a feedback loop to allow for incremental improvements of the automated transformation. Because of these limitations, these ideas cannot be considered a method fragment yet but these ideas can be developed into a more formal approach, and method fragment, to map DEMO models to NS elements (see Chapter 8).

Huysmans also notes that as the "organizational elements" are missing in the enterprise ontology, it is hard to identify combinatorial effects that hamper flexibility, and, as a result, that it is unclear to what extent combinatorial effects on the organizational level can or should be avoided. This strengthens the idea to

include enterprise implementation (Section 4.3) into the design of a method for Enterprise Model-driven Software Development.

5.2.4 Realization: From O to I to D

De Jong [102, 101] suggests an approach, called realization, to first create the ontological models of the infological and documental organization (see Section 4.2.3), based on the ontological model of the O-organization. In order to do so, he defines multiple transaction kinds for sharing, recalling, remembering and archiving P-facts to support a single original transaction kind. However, he does not seem to define transaction kinds for sharing or remembering C-facts.

De Jong recognizes that the ontological model alone is not enough to fully generate supporting software. Thus, in a next step de Jong suggests implementing the actor roles in all (O-, I-, and D-) organizations and for all (P- and C-)acts. Although he calls his approach a method, he mainly provides examples of implementation, but does not suggest a detailed framework that can be used to repeatedly decide on the implementation of acts and actor roles.

More specifically, table 11.1 in [124] shows a remember transaction for the object when dealing with a request. A critical review from an ontological perspective however yields that this is wrong: since this is the action rule for dealing with some act on the object, the object was already there and a share transaction would be the only logical choice. Possibly the choice for a remember transaction is the result from an implicit design decision where the executor of the transaction still needs to save the information in its own data storage.

Research on enterprise implementation has shown that implementing actor roles can be very complex and has its implications on software design and development (see Section 4.3). Moreover, in a world that is full of automation, the implementation is typically with IT, that is aimed at the infological and documental layer. The question then arises whether applying the realization approach is really necessary or over-engineering.

Supported by the statement that software can be created without the creation of the ontological models of the I- and D-organizations [117], using the layered organization approach does not seem necessary for this research. The realization approach is considered a potential method fragment that will not be further explored in this research.

5.2.5 DEMO CTP Engine

Van Kervel et al. [465, 466, 421] report on a software engine able to generate working operational software, taking the ontological enterprise model of a domain as input. As opposed to both Huysmans and de Jong, he believes that enterprise ontology is enough to create working software. A further investigation however revealed that in the process of creating this engine, several implicit (organization) implementation design decisions have been made and ended up being hard coded in the engine. Moreover, this approach only seems to support changes in the

ontological level, i.e., a new or changed product, service, or business rule, and not on the implementation level, i.e., a new functionary type or organizational unit.

At the same time, van Kervel has done extensive work in creating a state machine and a dynamic logic model to support the CTP (see Section 4.2.1), including the automatic evaluation of action rules. There are however comments that it is not fully compliant with the theory [179]. A quick analysis shows that indeed it is not compliant anymore with the latest version of the CTP as little, but important, details have changed over time.

Gouveia provides an alternative microservice based implementation of the CTP using pots [178], based on Normalized Systems theory. In his attempt to simplify the state machine, he seems to lose however some benefits of using enterprise ontology and the CTP. Also, he seems to mix enterprise ontology and implementation without a clear view on the separation of the two, making it impossible to make design decisions explicit, and possibly hampering agility.

Although both attempts have their limitations, there seems to be value in supporting the CTP by working software. Both works are identified as a potential method fragment and the focus for this research will not be in creating software to support the CTP.

5.3 Conclusions

In this chapter the concept of method fragment and its categorization are explored. In order to be able to create a (situational) method, several method fragments are needed.

Furthermore, existing research in the area of creating software from DEMO models is reviewed on the presence of usable method fragments. Although first attempts have been made to create software from Enterprise Ontology, the conclusion is that there is still a big gap to create a complete method to generate software from DEMO models that completely supports the end users and is flexible in its implementation to the level that is *required by the enterprise*. Most efforts seem to be put into creating software components to support the CTP, but in doing so many researchers seem to have skipped the step of explicitly addressing enterprise implementation, resulting in the need for developers to deal with such decisions themselves, possibly diminishing the end user support as well as the quality and evolvability of the resulting software.

In Table 5.1 the relevant method fragments as found in literature are classified. In particular, there seems to be a lack of technical fragments and a lack of support for all DEMO aspects models. It strengthens the idea that method fragments are needed that are *executable* by IT and cover *all* aspects of the DEMO metamodel (Section 4.2.5), involving different technologies, and explicitly addressing organization implementation decisions, that can later be combined into a (situational) method. This will be the basis for the practical research in Part II.

Abstraction / Granularity	Perspective: Product	Perspective: Process
Conceptual / abstract		
Method		GSDP
Stage		MDSD, DEMO2services, DEMO2components, DEMO2NS, realization
Model	DEMO aspect models, implementation model	OER
Diagram	DEMO diagrams and tables	
Concept	see DEMO metamodel (Figure 4.7), OIVs (Section 4.3)	
Conceptual / implementable		
Method		
Stage		
Model	DEMO exchange model (XML and JSON)	DEMO2WSBPEL
Diagram		
Concept		
Technical		
Method		
Stage		CTP engine
Model	OpenModeling, Simplified Modeling Platform, Sparx MDG	
Diagram		
Concept		

Table 5.1: Identified existing method fragments categorized

Part II

Exploratory Case Studies

Parts of this chapter are originally published as M. R. Krouwel and M. Op 't Land, *Business Driven Micro Service Design - An Enterprise Ontology based approach to API Specifications* [264].[1]

6 ECS 1: Specifying Microservices

Abstract. As technology is evolving rapidly and market demand is changing quicker than ever, many are trying to implement service orientation and adopt market standards to improve adaptivity. A microservice architecture makes applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features. The question then arises how to design a manageable and stable set of microservices that is sufficient for the business. This exploratory case study systematically deduces an algorithm to derive a set of microservices, expressed according to the OpenAPI standard, from the ontological model of an enterprise, that is stable by nature, sufficient for the business, and based on units of clear size. This algorithm has the DEMO actor cycle at its heart and has been evaluated with the real-world case Social Housing from ICTU by creating a SwaggerHub implementation. Further research should clarify the role of implementation choices in the algorithm.

6.1 Problem Formulation

As technology is evolving rapidly and market demand is changing quicker than ever, many enterprises are trying to implement service orientation and adopt market standards to improve adaptivity [111, 289]. However, when applying microservices at an industrial scale, the manageability of its underlying software service portfolio can become a problem [129]. Indeed, a large portfolio of "small" services enables high adaptivity – just as sand is much more flexible in the construction world than stones or prefab walls – but its governance can be a nightmare, low-

[1] A detailed mapping from the original paper to this thesis can be found on p. xxx ff.

ering its business value. This exploratory case study aims to find a method to design a manageable, complete and stable set of microservices that is sufficient for the business, while being adaptable in its software implementation.

As literature is limited on specifying the 'right' size of a microservice, or sometimes even in contradiction (see Section 3.4.2), in this research, a goal is to find a clear size for the microservice, that is sufficient for the business and independently changeable from a software perspective. Poorly defined microservices can lead to increased development costs and often miss important use cases entirely [209]. O'Reilly finds, in a survey with 1502 respondents, that decomposing (business) functions into microservices is one of the biggest challenges in adopting microservices [289]. In this research, a goal is to find a set of microservices that is stable and complete from a business perspective, thus with a traceable mapping from business (services) to microservices – as soon as an enterprise changes its business, e.g., from pigsty to restaurant, its portfolio of (business) products and services will change anyhow, including its supporting microservices.

This exploratory case study is inspired by Stichting ICT Uitvoeringsorganisatie (ICTU),[2] a Dutch organization that supports Dutch government by exploring methods, IT solutions and platforms that are built around the principle of continuous change in organization and technology. The Social Housing domain is of particular interest for ICTU as it's being reorganized every few years and many parties are involved. In order to create stability within this domain, independent of how it is organized, it is desirable to have a stable and complete set of technology-independent microservice specifications.

Summarizing, the goal is to find an algorithm to define a set of microservice specifications that:

C1 is **stable** from a business perspective with respect to a set of (business) products and services;

C2 is **complete** – i.e., sufficient and not more than strictly necessary – from a business perspective;

C3 is **changeable** in its internal software implementation, while the external behavior remains unchanged; and

C4 contains services of a **well-defined size** – i.e., with a clear scope.

This exploratory case study reports on an algorithm that defines the (traceable) mapping from an ontological enterprise model to API specifications. This chapter is structured as follows: the background for this chapter was introduced in Section 4.2, specifically the actor cycle, and Section 3.4.2; in Section 6.2 the created algorithm is shown, along with its evaluation with the real-world case 'Social Housing' from ICTU; in Section 6.3 the results and learnings of this exploratory case study are summarized and in Section 6.4 the learnings are formalized in terms of the goals within this research.

[2] https://www.ictu.nl/

6.2 Building, Intervention and Evaluation

As shown, earlier attempts have been made to define (micro)services from DEMO ontological enterprise models (see Section 5.2), because they *a)* are stable with respect to a set of (business) products and services, *b)* are complete with regard to all real-world business actions, and *c)* use the transaction notion to decouples transactors in a way very similar to SOA (see Section 4.2). In order to address criteria C1 and C2, and C3, DEMO models are used as input for the algorithm. As an API specification contains a stable method and protocol for communication while the technological implementation behind it remains unknown for the consumer of the API and thus can change (see Section 3.4.2), the microservices will be defined as APIs, in order to further address criterion C3. The created algorithm explicitly defines the (traceable) mapping from the enterprises ontological model to API specifications, making sure no use cases are left out and resulting in a clear scope, focused on a single task, and thus with a well-defined size, addressing criterion C4.

Earlier approaches seem to have ignored the actor cycle (Section 4.2.2) to define a set of (micro)services. As the actor cycle is key in order to support business actors, the algorithm in this research is mostly built around the actor cycle (Section 6.2.1). The algorithm has been evaluated on the Social Housing case from ICTU (Section 6.2.2), which resulted in many refinements. Below the final version of the algorithm is shown.

6.2.1 Deducing the Algorithm From the Actor Cycle

By deducing the algorithm from the actor cycle, completeness can be claimed from a process perspective: in the operation of an organization, actors typically select an agendum from a list of agenda and process the selected agendum by retrieving the applicable action rule, assessing the action rule, deciding on the response and performing new acts (see Figure 4.5). As DEMO itself claims comprehensiveness and conciseness (see Section 4.2), together this makes sure criterion C2 is met.

By taking apart the different steps in the operating cycle, several kinds of microservices can be defined.

1. *select agendum*: the services that provide the possible agenda for an actor, fulfilling an actor role, to select an agendum from, are called *agenda services*;

2. *get applicable action rule(s)*: in an environment that is supported by software, using the right services can be enforced by the specifications and no additional services are needed;

3. *assess*: the services that provide the information on how to proceed are called *assess services*, that will use *read services* that retrieve the facts;

4. *decide on response*: in order to comply to the autonomy of actors (see Section 4.2.2), the decision is left up to human actors and no services are defined for this step;

5. *perform act(s)*: the services that perform the act(s) are called *response services*, that will use *write services* to create the facts.

These services will be explained in more detail below.

Write Services Following the CTP, for each transaction kind 19 write services have to be defined, one for each C-act kind (18 per transaction kind) and one for the P-act, that deals with performing the act. For the API specification it is irrelevant who uses it in order to perform an act – that will be a matter of authorization. Additionally, write services are needed for all entity types in focus.

Read Services As information needs to be available for the actors, for each fact type in the DEMO FM (both OFD and all DFSs) a read service must be defined. The assess service for every action rule is considered a special kind of read service that uses or aggregates other read services.

Assess Services For each ARS in the AM, an assess service must be defined that evaluates the assess part of the action rule. As the assess part basically consists of mathematical expressions on facts, the assess service will use read services to retrieve the necessary information.

Response Services One response service is needed for each ARS, to allow actors to easily perform one or the other branch of the response part of the action rule. As the response part can consist of performing multiple acts, it will use one or more write service(s).

Agenda Services For each actor role, an agenda service is needed in order for the actor to gain insight in the agenda that require follow-up. It might be useful or necessary to create more detailed services to only retrieve the open agenda with a certain state (e.g., being promised), in order to be able to easily deal with delegations.

Complete Algorithm

The algorithm below shows how to transform a DEMO model into a set of (micro)service specifications cf. the API metamodel (Figure 3.6a). The transformation is visualized in Figure 6.1.

Input: Ontological model of the enterprise, covering all aspect models, expressed in DEMO-SL;

Output: Set of microservice API specifications;

Steps:

1. For each transaction kind in the CM: generate 19 write services (one for each C-act kind and one for the P-act) for performing the act, e.g.:

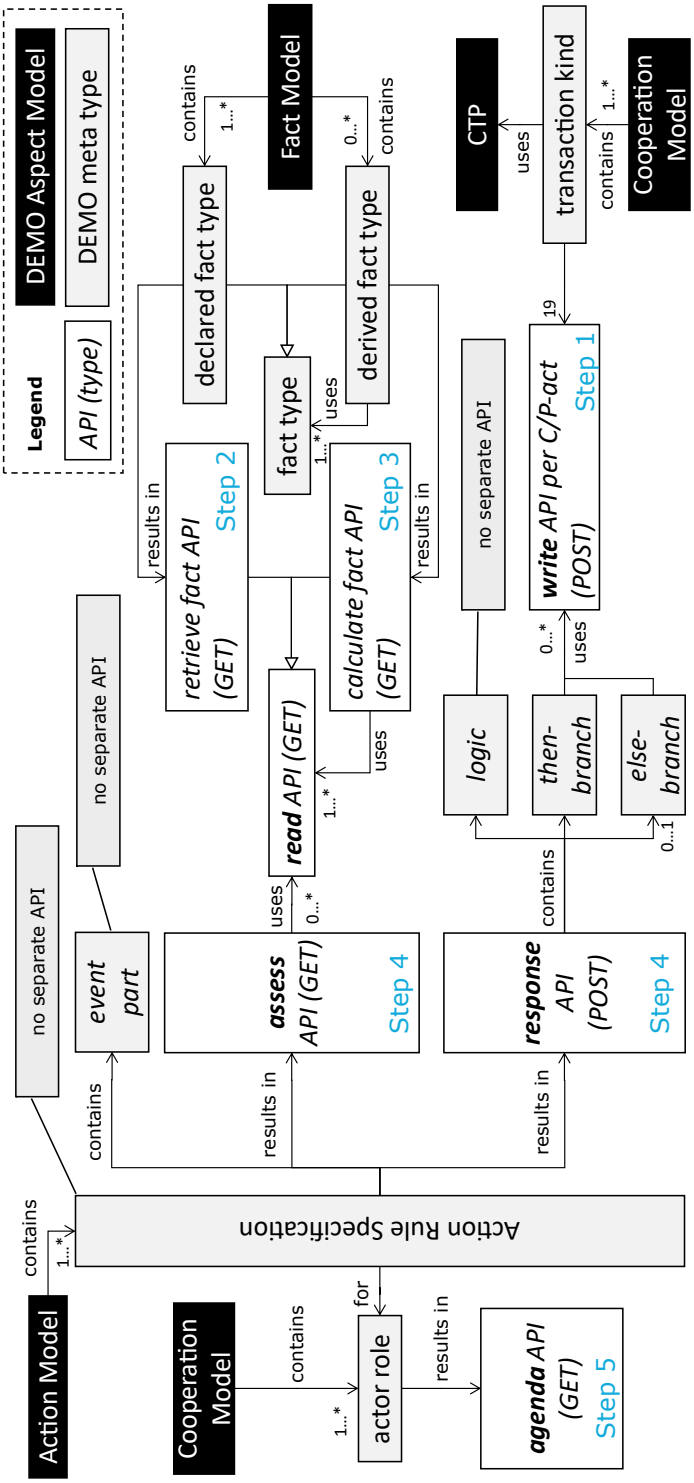


Figure 6.1: Visualization of the mapping of the DEMO metamodel to API specifications

Name: TK01request

Operation type: POST

Summary: creates a TK01request if all needed parameters are provided

Parameter(s): registration (the variable in the product kind)

Response(s): created agendum or error message if agendum could not be created

2. For each declared entity type in the OFD: generate one read service for retrieving the related data, and generate one write service if the entity type is in focus, e.g.:

   Name: registration

   Operation type: GET

   Summary: returns registration data for given ID if it exists

   Parameter(s): registrationID (integer)

   Response(s): registration data for given registrationID or error message if registration with registrationID does not exist

3. For each DFS (Derived Fact Specification): generate one read service for calculating the derived fact, e.g.:

   Name: calculateAge

   Operation type: GET

   Summary: calculates and returns the age for a given person if it exists

   Parameter(s): person, day (datetime)

   Response(s): age in years (integer) for given input

   This service will use other read services.

4. For each ARS in the AM: generate one assess service and one response service, possibly introducing additional services for several (sub) parts thereof, e.g.:

   Name: assessARS01

   Operation type: GET

   Summary: evaluates the assess part of the ARS for a given C-act if it exists

   Parameter(s): cactID (integer)

   Response(s): 'comply' or 'non-comply' (boolean) or error message if C-act with actID does not exist

   Name: responseARS01

   Operation type: POST

   Summary: Performs the response for the given C-act and decision

   Parameter(s): cact, decision (boolean)

   Response(s): list of created agenda that have been created as result of performing the response, or error message if C-act with act has already been dealt with

The assess service will use other read services while the response service will use write services.

5. For each (elementary and composite) actor role in the CM: generate one agenda service, e.g.:

   Name: agendaForAR01

   Operation type: GET

   Summary: returns a list of open agenda that AR01 needs to deal with

   Parameter(s): none, or possibly filters like the act kind or transaction kind

   Response(s): list of agenda to deal with for AR01, or empty list if such agenda cannot be found

For some services, object identifiers (IDs) were introduced, which is quite common in software development and database design. The algorithm does not generate PATCH, PUT or DELETE services, as from an ontological perspective information and data are never changed or deleted. The algorithm also does not generate OPTIONS, HEAD, TRACE, or CONNECT services as those operation types are more on the infrastructural level. The algorithm has been implemented in Python[3] to ease generation and validation of the service specifications.

6.2.2 Evaluating the Algorithm

As mentioned in Section 6.1, the Social Housing domain is of particular interest for ICTU as it's being reorganized every few years and many parties are involved. In order to create stability within this domain, independent of how it is organized, it is desirable to have a standard set of APIs specifications. Moreover, there is enough documentation available for this domain as the researchers have done more work in this area [342].

In the Social Housing domain two main areas can be discerned: *a)* the registration of a home-seeker as a member, and *b)* assigning a house to the member. The focus for the evaluation was on the first.

Input: Ontological Enterprise Model of the Social Housing Domain

The CSD (Figure 6.2) and TPT (Table 6.1) reveal the starting, periodic renewal and ending of a registration. Starting the registration is initiated by the (aspirant) member and executed after the registration fee has been paid. Every year the registration is renewed against payment of a renewal fee. Ending a registration can be initiated by the member – e.g., when moving to another area – or by the Social Housing organization – e.g., in case of (repeated) non-payment of the renewal fee. The model shows that actors in this domain need access to facts about costs and terms, and about persons and (their) living, abstracted from how access to these facts is arranged.

[3]Source code can be found at `https://github.com/mkrouwel/demo2api`.

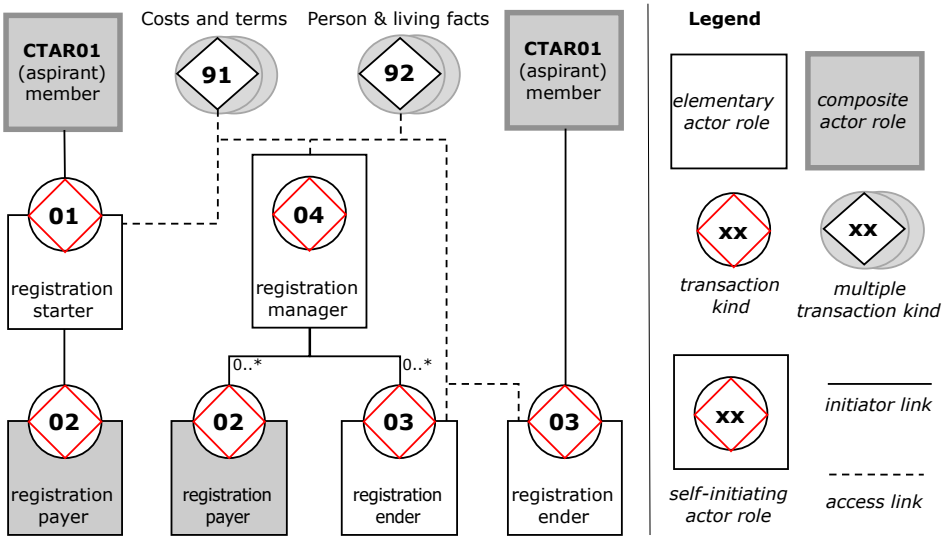


Figure 6.2: CSD for Social Housing

Transaction Kind	Product Kind	Executor Role
TK01 registration starting	PK01 [registration] **is** started	AR01 registration starter
TK02 registration paying	PK02 **the** fee **for** [registration] **in** [year] **is** paid	AR02 registration payer
TK03 registration ending	PK03 [registration] **is** ended	AR03 registration ender
TK04 registration management	PK04 registration management **for** [year] **is** done	AR04 registration manager

Table 6.1: TPT for Social Housing

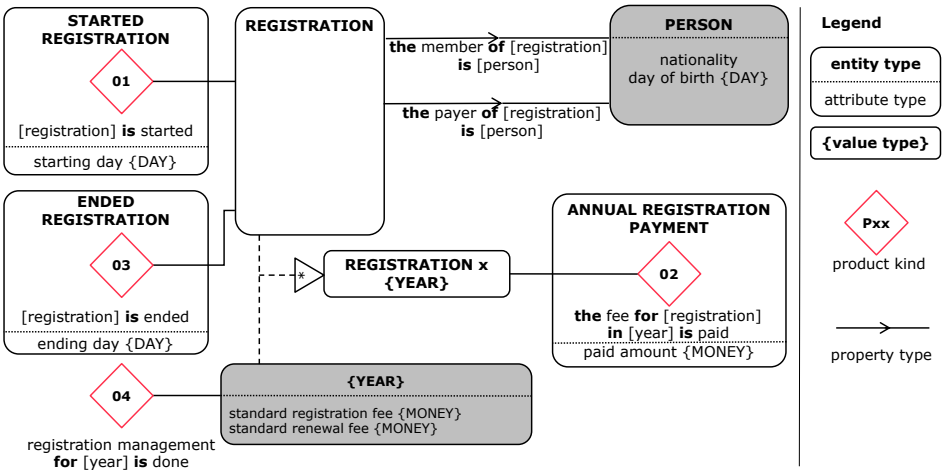


Figure 6.3: OFD for Social Housing

the age **of** [person] **on** [day] ≡ [day] **minus the** day of birth **of** [person]
[person] has active registration **on** [day] ≡ **there exists a** [registration] **for which the** member **of** [registration] **equals** [person] **and** [registration] **is** started **and** starting day **of** [registration] **is smaller than or equal to** [day] **and** (**NOT** [registration] **is** ended **or** ending day **of** [registration] **is greater than** [day])

Table 6.2: DFSs for Social Housing

The OFD (Figure 6.3) shows the REGISTRATION as core entity type, and the starting and ending of a registration as product kinds. Additionally, the entity type PERSON is shown – gray colored and thus out of focus, as persons are not created within this domain – including the property type that a person is the member and/or payer of a registration. The value type {YEAR} is included to express *a)* the product kind of annual registration payment, *b)* the definition of the (aggregated) entity type REGISTRATION × {YEAR} for the product kind 'annual registration payment', and *c)* several decisions taken yearly – modeled as attribute type of {YEAR} – such as the standard registration fee. In deciding upon starting a registration, (derived) facts about the existence of one or more active registration(s) for and the age of a person are needed; Table 6.2 shows the DFSs to calculate those.

Action rules guide actors in their decisions. Table 6.3 shows the ARS for the registration starter (AR01) to settle the agendum kind *registration starting is requested* (TK01/rq). This action rule says to assess that the participants are

event	**when** registration starting **for** [registration] **is** requested			(TK01/rq)
	with	**the** starting day **of** [registration] **is some** day; **the** member of [registration] **is some** person; **the** payer **of** [registration] **is some** person		
assess	**if**	*rightness:*	**the** performer **of the** request **is the** member **of** [registration]; **the** addressee **of the** request **is a** registration starter	
		sincerity:	** no specific condition **	
		truth:	**the** age **of** member **of** [registration] **on** starting day **of** [registration] **is greater than or equal to** 18; nationality **of** member **of** [registration] **is** Dutch; **NOT** member **of** [registration] has active registration **on the** starting day **of** [registration]; **the** year **of the** starting day **of** [registration] **is greater than** **or equal to the** year **of** Now	
response	**if**	*performing the action after* ***then*** *is considered justifiable*		
	then	promise	registration starting **for** [registration] **to the** performer **of the** request	[TK01/pm]
	else	decline	registration starting **for** [registration] **to the** performer **of the** request **with** ** reason for declining **	[TK01/dc]

Table 6.3: ARS01 for AR01 (TK01/rq)

authorized to play their (performer and addressee) role in this request, that the (aspirant) member is at least 18 years old and Dutch, that the (aspirant) member doesn't have an active registration at the starting day of the new registration, and that the starting day is in the current year or later – the latter implies that a registration can start retrospectively, but limited to the current year. If the assessment yields a positive result, normally the registration starter can proceed to request the (aspirant) member to promise that the registration will be started; otherwise the registration starter normally should decline to do so. As an action rule is not fully deterministic, the registration starter remains free to – responsibly! – deviate from this rule. For the chosen focus within the Social Housing domain, 11 ARSs have been defined (see Appendix B).

All DEMO models have been converted (manually) into a JSON format (see Appendix C) that can easily be fed into the reference implementation. In order to be able to generate working software, the only extension made to the DEMO model was a mapping from value types and result types of derivations to primitive software data types such as number, boolean, or datetime.

Output: List of API Specifications

In Table 6.4 for each step in the algorithm it is shown which services are created. The complete generated YAML file for the given input can be found in Appendix D. All (109) services have been loaded into SwaggerHub,[4] a tool for API design and documentation based on OAS, to confirm that the output of the algorithm conforms to OAS. As an example, the specifications for two services are shown in Figure 6.4.

Step	Microservices	Total	Type
1.	TK01request, TK01promise, TK01decline, ...	$4 * 19 = 76$	POST
2.	registration, person, year	3	GET
		1	POST
3.	calculatePersonAge, calculatePersonHasActiveRegistration	2	GET
4.	assessARS01, responseARS01, assessARS02, ...	11	GET
		11	POST
5.	agendaForCTAR01, agendaForAR01, agendaForAR02, agendaForAR03, agendaForAR04	5	GET
Total		109	

Table 6.4: Full list of APIs for Social Housing as generated by the algorithm

[4] https://swagger.io/tools/swaggerhub/

```
paths:
 /TK01rq:
  post:
   summary: creates a new TK01-rq
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/StartedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input

 /agendaForAR01:
  get:
   summary: retrieves agenda for AR01
   responses:
    '200':
     description: agenda for AR01 retrieved
     content:
      application\json:
       schema:
        type: array
        items:
         $ref: '#/components/schemas/CFact'

components:
 schemas:
  CFact:
   type: object
   properties:
    performer:
     type: string
     example: 'Martin'
    addressee:
     type: string
     example: 'Erik'
    intention:
     type: string
     example: rq
    product:
     $ref: '#/components/schemas/ProductKind'
```

Figure 6.4: Specification of services TK01request and agendaForAR01, and component CFact, in OAS format

During development, it was found that defining the DEMO product kinds, entity types and value kinds as reusable specification components greatly reduced the complexity of the specifications; in OAS these are defined as components. Additionally, a reusable (specification) component for *CFact* was developed, shown at the bottom of Figure 6.4. This component can easily be used for other cases as well. It was chosen to use objects instead of object identifiers where possible, a decision that is easily changed as the services are there to get an object by its identifier (ID).

6.3 Reflection and Learning

In this section a reflection is provided whether the algorithm meets the goals and criteria that were set in Section 6.1 and on the design decisions that were taken in the creation of the algorithm.

6.3.1 Reflecting on the Criteria

The algorithm for specifying microservice APIs will be evaluated with respect to the criteria as defined in Section 6.1. As the set is generated from an ontological model of the enterprise, it's by definition stable with respect to a set of products and services, meeting criterion C1. As new products or services are identified, the algorithm can easily generate the additional microservices. As ontological models are said to be comprehensive and concise, and the operating cycle is used as a basis for the algorithm, the resulting set is complete, meeting criterion C2. Organization implementation [265], such as delegations, might require additional services or additional parameters in existing services. As the set only specifies APIs for the microservice, their software implementation is undecided yet and thus changeable, meeting criterion C3. In fact, meeting criteria C1, C2 and C3 follows directly from the deduction of the algorithm from the actor cycle.

Criterion C4 can only be reflected on empirically. It is claimed that all microservices are well-defined and are focused on one (business) responsibility, possibly using other services, and therefore have a clear scope and size. Although an externally measurable size for a microservice is not provided, this research at least progressed the definition of the 'right' size for a microservice. Generating service definitions on this level makes it easier to maintain the large amount of services – this concept is similar to that of Normalized Systems expanders (see Section 3.4.3) – even by multiple and distributed teams. Additional research is needed to define a method to measure the size of a microservice, as well as to define the 'right' size for a microservice, possibly depending on many variables.

The algorithm mostly produces GET (read) services on the 'raw' data (such as registration) and only provides POST (write) services on the process level, leaving out POST services on most entity types and completely leaving out services of another operation type, such as PUT, PATCH and DELETE. This is in contrast to current practices that define services of (almost) all HTTP operation types on the data level and usually leaves out services on the process level. Taking the C-act as unit of work – embedding one independent and several dependent facts

– provides integrity from a process perspective, ensuring that only a consistent set of data can come into existence. As a result, the algorithm creates services that are driven by business (demand), where current practice typically seems to define microservices from a supplier perspective on top of existing data sources.

During the evaluation of the algorithm, it was found useful to validate the meaningfulness for the business end users. As it wasn't part of the criteria, this validation was not done. In further evaluations it seems necessary to do additional field evaluation with end users.

6.3.2 Reflecting on the Design Decisions

During the application of the algorithm, a mapping from the value types in the OFD, i.e., year, day, money, nationality, to data types had to be made. Currently this is done by extending the DEMO model with primitives. It might be desirable to separate this mapping to primitives from the input model in a next iteration.

While HTTP is one of the more popular internet protocols, others exist, such as SMTP, (S)FTP, MQTT, AMQP and XMPP. A next iteration of the algorithm can include extensions or abstractions to other protocols, especially where there are relevant standards available, similar to OAS for HTTP.

Currently, the decision part – the first and default sentence of the response part – of an action rule is left out of the design of APIs. It could be argued that this can also be seen as a microservice, though often – or at least: preferably – executed by a human actor. Adding such a decision API would also create the opportunity to add an action rule API, covering the complete execution of a single action rule (steps 2-5 of the actor cycle), thereby making it possible to completely outsource its implementation – as part of further research on the mutual dependence of organization splitting with IT splitting [339].

Instead of creating one generic service for handling C-acts, it was chosen to define a set of services. The reason for doing so is to minimize complexity in the implementation and to adhere to the principle of 'Separation of Concerns'. By separating this service into many smaller services, it also becomes easier to grasp the advantages of (true) microservices such as independent scaling.

Some transformation steps in the algorithm are or can be influenced by (organization and/or software) implementation choices (see Section 4.3), such as regulations regarding archiving or security, delegation of work from one actor to another, the mapping from value types to technical (primary) data types, as well as *how* external information is provided – e.g., completely given by the initiator or referenced by with a key, including the permission for the executor to use the key to retrieve the required information. The role of such implementation choices should be made more explicit in next iterations of the algorithm, possibly resulting in additional services.

In the algorithm read services are defined on the entity and value type level. However, it could be considered to further detail this into different microservices on the level of attribute type, property type and product kind, supporting an even more distributed data storage model and possibly a better performance as only the strictly required data is retrieved. The downside however can be that splitting

this up into several services introduces an overhead in computing and network latency with a lower performance as result. Further research should investigate whether splitting up to a lower level is useful.

6.4 Formalization of Learnings

In this exploratory case study the goal was to deduce an algorithm to generate a set of microservice specifications that is stable, complete, changeable in implementation and contains services of well-defined size. It has been shown that DEMO offers a good starting point to generate microservices that are (relatively) stable and (claimed) complete. By defining microservices in terms of APIs, they are changeable in their implementation. As the algorithm generates a fine-grained set of microservice specifications that are all focused on one (business) responsibility, it is claimed they have a well-defined scope and size.

This exploratory case study has shown that defining (implementable) (micro)services specifications from DEMO models is possible. The only implementation details added are the (software) primitives for the value types in the OFD. This also shows that software can be built around a steady core while being flexible in its software implementation. Using the realization approach (see Section 5.2.4) does not seem necessary, nor does it address the issue of specifying primitive types. This exploratory case study confirms that the overall method should use DEMO models and (conscious) chosen implementation decisions.

6.4.1 Identified Fragments

The conceptual mapping from DEMO to microservice or API specifications is considered a conceptual-implementable process fragment. Its counterpart implementation in Python to generate API specifications in OAS from a DEMO model in JSON format is a technical process fragment.

6.4.2 Implications For Future Research

While generating API specifications can be useful, the software that conforms to these specifications still has to be built. As the specification is not an actual implementation, it makes sense that this exploratory case study did not include the concept of Enterprise Implementation. However, in order to really bridge the gap from conceptual enterprise models to software implementation, follow-up research should focus more on actual implementation in software, supporting concepts from the Enterprise Implementation Framework.

Parts of this chapter are originally published as M. R. Krouwel and M. Op 't Land, *Using Enterprise Ontology as a basis for Requirements for Cross-Organizationally Usable Applications* [263].[1]

7

ECS 2: Using Mockups to Validate Requirements for Similar Enterprises

Abstract. Developing software that can be (re)used by multiple (similar) enterprises is becoming increasingly important. However, actually reusing business software has been hindered in practice often by implicit assumptions about enterprise and software implementation. A procedure is proposed to gather and validate requirements for software for similar enterprises that, taking DEMO as starting point, *a)* is easy to communicate, *b)* can discern differences between similar enterprises, and *c)* has an attractive ROME. While evaluating this procedure in a large real-world case, it also appeared to be possible to *a)* systematically design the 'unhappy' flow as well, *b)* define software architecture principles for supporting enterprise agility, and *c)* systematically design mockup screens based on DEMO models. Finally, the procedure greatly objectified the discussions between all stakeholders involved.

7.1 Problem Formulation

Developing software that can be (re)used by multiple enterprises is becoming increasingly important [91, 496, 244, 19]. Especially in (international) cooperation(s), ranging from open-ended without any agreements on one side to one with a high degree of sector agreements and standards on the other side, reusing (business) software is becoming more and more popular [243]. Being able to develop a piece of software once and use it in several enterprises is, obviously, one of the

[1] A detailed mapping from the original paper to this thesis can be found on p. xxx ff.

main benefits to be achieved.

Reusing IT components among multiple enterprises is a great concept in theory and has been quite successful in the context of inter-enterprise reuse of technical infrastructure: modern operating systems, database management systems, component infrastructures, web servers, web browsers, etc., are examples of infrastructural components that can be (re)used by multiple enterprises [54]. Reusing business software, however, has been hindered in practice often by implicit assumptions about enterprise and software implementation [494, 458]. How often is similar software developed multiple times, just because of small differences in the requirements? And how often do enterprises adapt their processes to fit a standard software package, such as an ERP system, as it is not possible to adapt the package to meet the enterprise's specific needs?

This exploratory case study reports on a procedure that uses DEMO (see Section 4.2) and mockups in the process of gathering and validating requirements for software that can be used by similar enterprises. The procedure *a)* is easy to communicate, *b)* can discern differences between the (similar) enterprises, and *c)* has an attractive Return On Modeling Effort (ROME). By using mockups to elicit the requirements and validate enterprise models, within a relatively short period great insights in the similarities and differences between enterprises and the consequences for their supporting software can be gained.

The procedure has been evaluated in a large real-world case at an International Public Private Organization (IPPO)[2] that supports common concerns of National Offices that grant patents. These National Offices have their own way of working, including their own supporting software systems that were loosely based on what once were copies of the same software system; several offices have adapted the software in terms of business rules, supported processes, and integration(s) with CMSs and DMSs. The goal of this exploratory case study was to find the differences between three National Offices to see whether these offices could be supported by one software system.

The remainder of this chapter is structured as follows: the background for this chapter was introduced in Section 3.2 (GSDP), Section 4.2 (DEMO) and Section 3.4.1 (mockups); in Section 7.2 the concept of *similar enterprise* is detailed, the created procedure is presented, and the evaluation with the real-world case IPPO is shown; in Section 7.3 the results and learnings of this exploratory case study are summarized and in Section 7.4 the learnings are formalized in terms of the goals within this research.

7.2 Building, Intervention and Evaluation

First, the concept of similar enterprises is defined and principles and choices in the design of the procedure are explained. Then the procedure to gather and validate requirements for the design of software of similar enterprises is elaborated. The desired end result of this procedure is a validated set of business and

[2]Between 2012 and 2022, 6,000 to 7,000 employees worked at IPPO in over 30 countries.

software requirements that makes explicit the similarities and differences between the investigated enterprises.

7.2.1 Similar Enterprises

Enterprises are said to be similar when they offer the same kind(s) of products and/or services. For example, all health care insurance companies are considered similar, as they all provide health care insurance, although the exact products delivered may vary. In terms of GSDP, enterprises can be said to be similar when their businesses share the same function model. Multiple enterprises sharing the same function model can have different enterprise ontologies. And when those enterprises have the same enterprise ontology, their enterprise implementations are likely to differ. In an ideal situation, one piece of software supports similar enterprises, possibly with (slightly) different enterprise ontologies and likely with different enterprise implementations.

In traditional requirements engineering processes, the resulting set of requirements usually describes the current or future enterprise implementation. However, this approach easily misses future changes in the enterprise and thus endangers reusing the supporting software in other enterprises.

DEMO has shown to offer a quick way to find similarities and differences on the enterprise ontological level, and to clearly distinguish those from similarities and differences in their implementations. For example, Op 't Land et al. [338] showed in six weeks that the enterprise ontologies of the Cargo business processes of the two merging parts of Air France and KLM were very alike and allowed to compare the enterprise and software implementation of these processes. Mulder [324] has looked at 22 complaints boards of the Dutch Stichting Geschillencommissies Consumentenzaken (Dispute Settlement Boards for Consumer Affairs) using DEMO and found that the 22 boards share the same enterprise ontology, but have different enterprise implementations.

7.2.2 Working Principles and Design Choices

The main principles to create the procedure were that

a) it should allow for open discussion,

b) it should be possible to involve employees with different positions, i.e., management, subject-matter expert, legal, etc., and

c) the models created are understandable and/or falsifiable by means of understandable views by everyone involved.

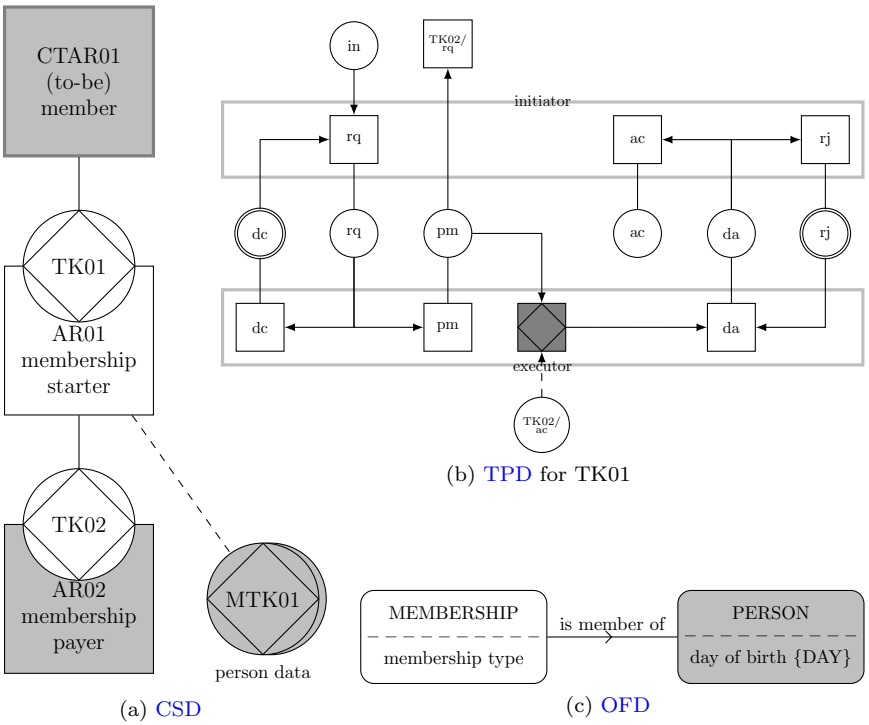
As DEMO allows to easily find similarities and differences between enterprises, DEMO models will be used as starting point. As communicating and validating DEMO models in their pure form is considered difficult (see Section 4.1.3), mockups will be used instead as they have proven to be understandable by most audience (see Section 3.4.1).

7.2.3 Procedure

The procedure is split into a preparation phase and a validation phase. The total time spent was 200 person days, of which 128 days were by Capgemini and the other 72 were by IPPO and National Offices.

Preparation Phase

In this phase, a draft DEMO model is created using existing materials such as process models and software implementation, possibly complemented with interviews.[3] In line with earlier research on transforming DEMO acts to software design [118, 219, 262], for each C-act agendum kind a mockup is created. As the being executed of a P-act can only be known by performing the corresponding declare C-act [119, p. 128], a separate mockup for performing the P-acts is not necessary. Depending on the act kind and its relevant action rule(s), the mockup should show the relevant data so that the actor can take a decision and/or enter new data.



(a) CSD

(b) TPD for TK01

(c) OFD

Figure 7.1: Selection of diagrams of the DEMO model for a fictitious membership office

[3] This is also known as *reverse engineering*, as shown in Figure 3.3.

Transaction Kind	**Product Kind**	**Executor Role**
TK01 membership starting	PK01 [membership] **is** started	AR01 membership starter
TK02 membership payment	PK02 [membership] **is** paid	AR02 membership payer

Table 7.1: TPT for a fictitious membership office

the age **of** [person] **on** [day] ≡ [day] **minus the** day of birth **of** [person]

Table 7.2: DFS for a fictitious membership office

The process of determining the contents of a mockup is shown by two examples using a fictitious membership case (see Figure 7.1, Table 7.1, Table 7.2 and Table 7.3). Suppose a transaction of kind TK01 is requested. It is of no relevance whether it is the member-to-be itself who enters the data into some (online) form, or whether it is an administrative employee who enters the data on behalf of the member-to-be. The data needed for the membership starting concerns some person data (see Figure 7.1c). If the person is not yet known in the database, this data must be entered – when the membership office uses the personal data as known by the government (residential registration office), every person is re-

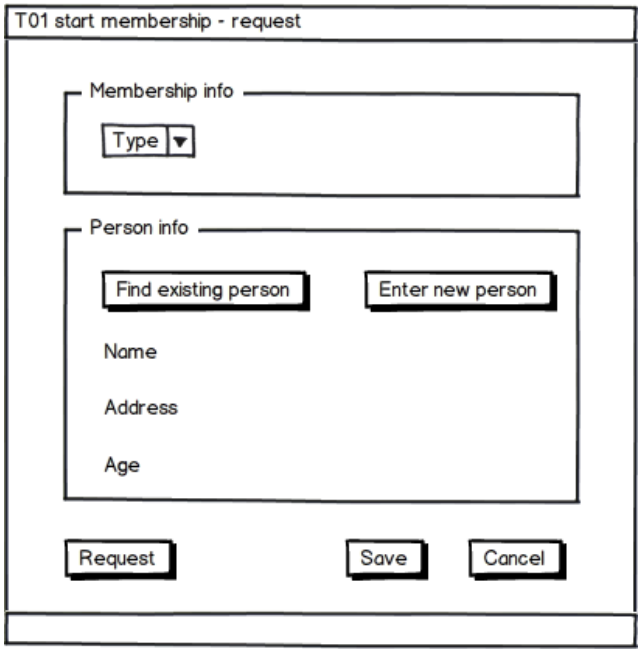


Figure 7.2: Example mockup to perform a TK01 request for the fictitious membership office

event	**when** membership starting **for** [membership] **is** requested		(TK01/rq)
	with	**the** type **of** [membership] **is some** membership type;	
		the member of [membership] **is some** person	
assess	**if**	*rightness:* **the** performer **of the** request **is the** member **of** [membership];	
		the addressee **of the** request **is a** membership starter	
		sincerity: * *no specific condition* *	
		truth: **the** age **of** member **of** [membership] **on** Now	
		is greater than or equal to 18;	
		there are no open payments (TK02) for any membership of which the member of the new membership is also the member	
response	**if**	*performing the action after* ***then*** *is considered justifiable*	
	then promise	membership starting **for** [membership]	[TK01/pm]
		to the performer **of the** request	
	else decline	membership starting **for** [membership]	[TK01/dc]
		to the performer **of the** request	
		with * *reason for declining* *	

Table 7.3: ARS01 for AR01 (TK01/rq) of a fictitious membership office

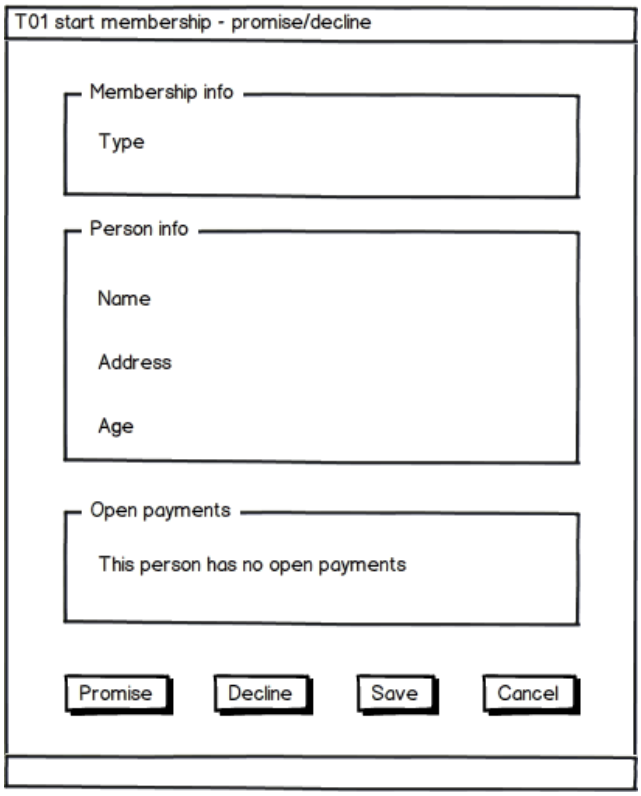


Figure 7.3: Mockup to respond to a TK01 request for the membership office

garded to be known. The resulting mockup (Figure 7.2) thus contains a part to enter data about the membership, a part to select or enter data about the member (person), and a button to perform the request, as well as to save (without submission) the current state of the form or to cancel and close the current form.

After being requested, actor AR01 now needs to decide on a promise or decline. In order to do so, it needs to evaluate the business rule as specified in Table 7.3. As a member (to-be) can have multiple memberships at the same time, the membership office wants to check there are no open payments for any membership here the member of the new membership is also the member. Thus, AR01 needs information about the membership, about the person and about the (open) payments. The resulting mockup (Figure 7.3) thus contains parts that show information about the membership, about the person, and about the (open) payments. The mockup also contains the buttons to decide between a promise and a decline, and additional buttons are provided for saving and canceling entered data, supporting a possible interruption in the operation.

Validation Phase

In this phase, the (draft) DEMO model and mockups are used to validate the requirements while enriching them with implementation details. The procedure defines the following steps that are performed for each business process that consists of a set of related transaction kinds:

1. Identify and validate the transactions kinds in the CM that are needed to create the product kind(s). This step is independent of a particular implementation, such as the working order, the way in which it is executed by human beings, and the way in which it is supported by software.

2. Identify and validate the acts in the PM and the dependencies. Make sure to distinguish true (existential) dependencies from a chosen order of working (implementation). As the PM is based on the CTP that provides not only the happy flow, the model now also include the 'unhappy' flow.

3. For each C-act from the acpm, determine whether it is performed explicitly or implicitly (tacitly). For explicit C-acts that have to be supported by software, the software must provide one (or more) screen(s) that shows the relevant data and allows for entering additional data. For implicit C-acts, no screens are needed.

4. From this step the created mockups are used to further validate the DEMO model. By showing the mockups for the explicit C-acts in the order as defined by the PM and possible chosen order of working, all findings from the previous steps are validated and/or improved upon. In particular, the preferred order of working will become more clear, consciously restricting the freedom left by the dependencies between acts as defined in the PM.

5. Discuss the contents of the mock-up:

- What decision is made: this should reflect the assess part of the relevant ARS for the C-act, and thus of the path or paths in the PM.
- What data is entered: this should reflect the response part of the relevant ARS for the C-act. Moreover, this data should be represented in the FM.
- What data is needed: this should reflect parts of the event and assess parts of the relevant ARS for the C-act. Moreover, this data should be represented in the FM, and the need for this data should be visualized in the CM by an access link.

For reference data, such as a list of countries or (yearly) fees, it should be discussed whether these lists should be manageable in the running software.

6. Discuss the applicable business rules per mock-up. This will validate the contents of the assess part of the relevant ARS for the C-act, providing input for the data needs.

7. For each mockup, discuss which department or functionary type uses it. This will make explicit how authorization and delegation is arranged.

8. Make sure to perform a consistency check between all the deliverables produced. Especially when transaction kinds are reused, such as payment, variations could have been introduced in the requirements for the data entered, needed, or the actor role assignment, depending on the business process at hand. The audience should be confronted with these differences, making explicit why these differences are there or concluding there are in fact no differences and the execution should be the same, no matter who initiated the transaction kind.

For these steps it doesn't matter whether the audience is from a single department, an entire enterprise, or multiple (similar) enterprises at the same time. Especially the latter could result in interesting discussions about the way the enterprise ontology is implemented. When a separate session is used for each enterprise or department, it is up to the facilitators to detect, and possibly discuss, the differences in implementation.

7.2.4 Evaluating the Procedure

Directed by IPPO, from 2002 to 2005 software was developed for the National Offices to support their business processes. This piece of software was highly adaptable in order to conform to national laws and differences in way of working, and, as a result, every National Office had its own software configuration.

In order to support a single request for an international patent, operationalized as separate patents in multiple countries, National Offices hand-over requests to other National Offices. In 2009, the idea came up to connect the systems to exchange data but an independent feasibility study in 2010 identified 54 problems on both the business – e.g., missing functionality, difficult interaction design –

and technical level - e.g., software failure, bad database design, and missing documentation. The causes of these problems were traced back to its original design: *a)* the design was based on an older version of the software reflecting a limited set of requirements, *b)* the National Offices were not involved in the design phase, and, *c)* as a consequence of the bad design and the high degree of customization, every National Office had created its own instance of the software and integrated it with local software such as a CMS and/or DMS.

In order to gather the requirements of these National Offices, an initial study was performed at three National Offices, making explicit the similarities and differences between them. The focus was on identifying the as-is situation at the business level, i.e., the main processes and the way in which they should be supported by software. The identification of optimization opportunities at the business level (to-be) was out of scope.

Preparation Phase

In the first phase, for each National Office a one-day workshop and a few interviews were held to gather the input to create the (draft) DEMO CM and PM. More than 90% of the defined transaction kinds were present in all three National Offices' CMs. Additionally, a (draft) FM was created by reverse engineering the existing database model, which was similar for all three National Offices. Based on the draft DEMO model, for every (relevant) C-act a mockup was created using a rapid wire framing tool (see Figure 7.4 for an example).

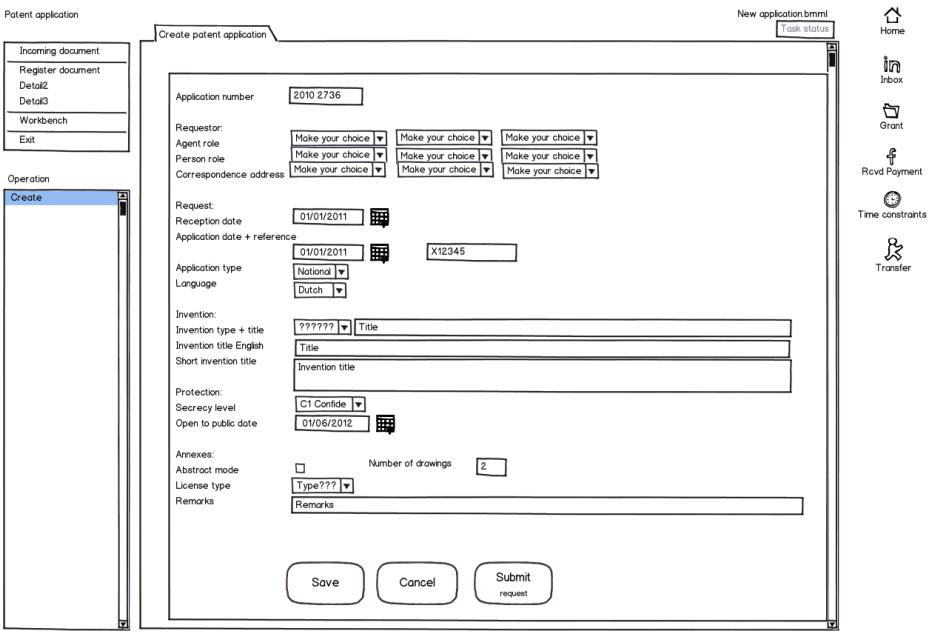


Figure 7.4: Example mockup for patent requesting

Validation Phase

In this phase, for three National Offices, a separate workshop[4] was organized in which the (8) steps mentioned above were executed. Each workshop lasted three days to discuss all business processes in an elaborate way with domain experts, IT directors, administrative, financial, and legal officers, and the sponsors of the project. For each step, large posters and slides were used in an interactive way (Figure 7.5). During the workshops, two team members made notes of the discussed topics, including relevant implementation details (Table 7.4) After each 3-day workshop session, the results were summarized and the notes were added to the DEMO models. To minimize interpretation issues, the notes and models were sent to the National Offices that were offered the opportunity to provide feedback in case notes were misinterpreted.

In the second and third workshops, insights gained from the previous one(s) were taken along and discussed, making explicit the differences between these National Offices. After having finished the three workshops, the similarities and differences between the DEMO models and implementation details of the three National Offices were consolidated. Specific results include:

- A DEMO model that fitted all three National Offices was created, detailing which activities were (not) applicable per National Office. The total CM (partly shown in Figure 7.6) consists of 70 transactions and 21 information links. Although there are almost no international regulations on granting a patent, the models are similar to a large extent. Some business processes or individual transaction kinds simply did not exist for some National Office(s), indicated by a gray diamond in Figure 7.6. For example, for National

Transaction kind	**C-act**	**National Office**		
		A	*B*	*C*
T01 - Grant patent	rq	E	E	E
	pm	E	E	E
	da	I	E	E
	ac	I	I	I
	dc	E	E	E
	rj	E	E	E
T07 - Examine formally	rq	E	E	E
	pm	I	I	I
	da	E	E	E
	ac	I	I	E
	dc	I	E	E
	rj	E	E	I

Table 7.4: Excerpt of the results of the implicit (I)/explicit (E) analysis

[4]Separate due to travel and language reasons.

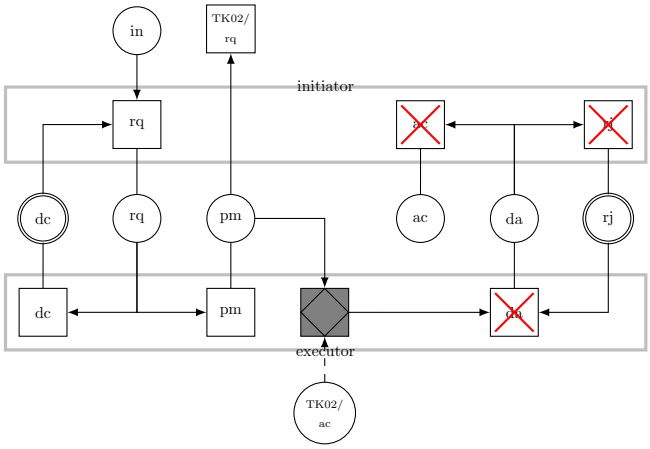


Figure 7.5: Example of capturing implicit/explicit

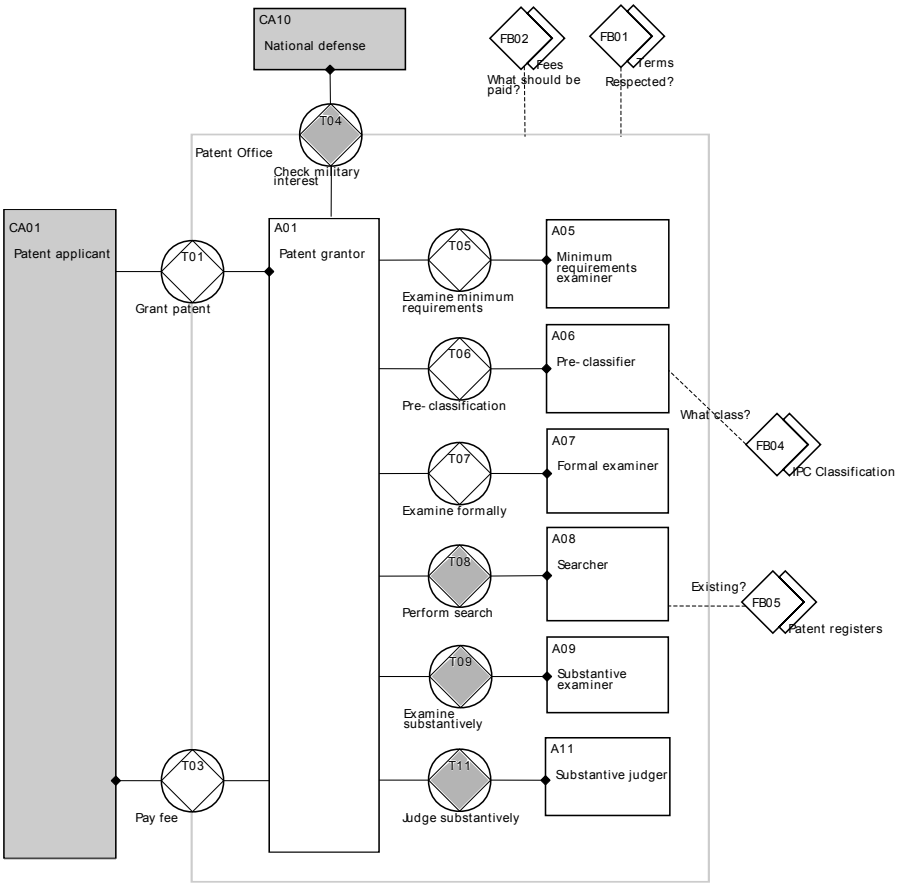


Figure 7.6: DEMO CM of patent granting

Office A *check military interest* (TK04) is mandatory by law, for National Office C it's optional, depending on the content of the patent requested, while National Office B never checks this. Also, National Office B does not perform a substantive examination or search, as a patent is always granted – in case of conflicting patents, the involved parties can resolve this in court.

- Although the true dependencies between transaction kinds are the same for every National Office, the specific order of working differed a lot. For example, while one National Office performed the search at the beginning to prevent unnecessary work, another one performs it at the end because the costs are high. This implies that different screen flows better fit the needs of the different enterprises.
- Whether an act is performed implicitly or explicitly is to a large extent the same for every National Office (Table 7.4). However, differences were also detected, especially in the transaction kinds that are both initiated and executed within the enterprise itself, i.e., there is no external party involved. For example, in National Office A it is not possible to decline any internally initiated transaction. And, while in National Offices A and B an implicit accept holds until an explicit reject is performed, in National Office C the accept has to be performed explicitly, otherwise a reject is assumed. The differences imply that not every screen is used by every National Office.
- The data entered at each screen (if applicable) for every National Office is the same. Also, the information needed (indicated by information links in the CM) is the same. The screens were very helpful as they brought the models alive.
- It was discovered the functionary type to actor role mapping differed a lot per National Office. Discussions about the fulfillment of actor roles at one National Office actually resulted them to introduce a new functionary type to fulfill actor role AR01 as coordinator of the grant process. By fulfilling this actor role explicitly, the control in coordination is expected to increase, shortening throughput times.
- As already indicated by some examples above, the sharing of information of other National Office helped them find the core processes and activities, as well as finding new business improvement opportunities.
- Based on the findings outlined above, two (software) architecture principles were defined:

  1. the software system should be able to support different orders of working (as a result of Step 4.); and
  2. the software system should be able to deal with different actor role to functionary type mappings (as a result of Step 7.).

The results were supported by all three National Offices and therefore the conclusion is that the procedure indeed helped to objectify the discussion. Based on these results, the estimation is that the differences between the three National

Offices, and probably between all National Offices, are relatively small, suggesting it is possible to create one piece of software to support them all.

7.3 Reflection and Learning

The goal of this exploratory case study was to design a procedure to gather and validate requirements for software for similar enterprises that *a)* is easy to communicate, *b)* can discern differences between the enterprises, and *c)* has an attractive ROME. In the validation phase, mockups were used to visualize the DEMO models that represent the functions of the software. The participants stated that the use of mockups was a great way to communicate and validate the, sometimes rather abstract, DEMO models, as they 'brought alive the system'.

The use of DEMO has shown its benefits in finding differences in both the enterprise ontologies and the enterprise implementations. Differences in the ontologies were mainly a result from differences in applicable regulations, such as the military interest check that was enforced by law for National Office A but not applicable for National Office B. The implicit/explicit analysis and actor role assignment analysis, as well as discussions about the screen flows, have offered great insights in the enterprise implementations, and helped in validating the enterprise ontologies. DEMO helped to distinguish the differences in the enterprise ontologies from differences in the enterprise implementations. While this exploratory case study was performed prior to the existence of the EIF (Section 4.3), in retrospect it can be concluded that the OIVs used in this procedure are *functionary type, order of working*, and whether an agendum kind is performed *explicitly or implicitly.*

With a total of 200 days spent for the analysis of 3 enterprises, including a consolidation of the similarities and differences, the ROME seems good. However, as there are no other cases studies that have used the designed procedure, and there are no similar approaches known, it is impossible to compare the ROME of this procedure to others.

In applying the procedure, some other interesting benefits were found: first, as the CTP includes the 'unhappy' flow, it is automatically a systematic part of the requirements and design of the software; second, (software) architecture principles were defined; and third, it has been shown that it is possible to systematically design mockups from DEMO models.

Using the CTP, it is possible to design screens independent of its actual user, i.e., whether the user is an internal employee or an external customer or supplier. The screens themselves are actually used for performing an (in)formative act, fully determined by the ontological act. The performer of the (in)formative act can but need not be the same as the performer of the ontological. For example, the screen used for requesting a membership (Figure 7.2) is for entering the data necessary for the membership as determined in the ontological act, i.e., member name, day of birth, etc. The same screen can either be used by both a member-to-be requesting the membership online and by an employee entering the data after a letter of request has been received as well as by the employee listening to

the member-to-be passing by in the office.

The IPPO case demonstrates that comparable enterprises, i.e., National Offices sharing the same function model, can have different ontological models, as the different enterprises are subject to different national laws and regulations. This also implies that DEMO models can change as these laws and regulations change over time. Since ontological changes often have a deep impact on supporting software, it is important to increase the insight in other aspects that can influence the ontological models of an enterprise.

The IPPO case also demonstrates that even when enterprises share the same ontological model, their enterprise implementations can differ significantly, also over time. Two (software) architecture principles were defined that could contribute to the possibility of one piece of software that supports different enterprise implementations of the same enterprise ontology.

As DEMO models are abstracted from implementation, it is inevitable not to introduce implementation decisions in designing mockups. For example, external fact banks have been considered to be part of the software as well. It could be decided to use external data sources such as the database of the residential registration office. It is interesting to see how additional software architecture principles can support such changes.

7.4 Formalization of Learnings

In this exploratory case study the aim was to find a procedure to gather and validate requirements for software for similar enterprises. As DEMO models have shown to support the easy finding of similarities and differences between enterprises, these were used as a starting point. As mockups have shown to improve speed and quality of the requirements engineering process and to be easily understandable by most audience, mockups are used in the procedure to communicate (parts of) the DEMO model. This exploratory case study has shown that it is possible to procedurally create mockups from a (draft) DEMO model, in order to validate and extend it with (relevant) enterprise implementation choices, such as the order of working, whether a C-act is performed explicitly or implicitly, and the functionary types that perform these acts. Moreover, the procedure seems to be effective and has an attractive ROME, as shown in validation with three National Offices from IPPO.

This exploratory case study strengthens the idea that software can be built around a stable core, but needs to be adaptable in its IT implementation to support changes in the enterprise implementation. At the same time, more research is needed to get a view on the complexity of supporting a flexible enterprise implementation with software. This exploratory case study confirms that the overall method should use DEMO and (conscious) chosen implementation decisions.

7.4.1 Identified Fragments

The procedure to create mockups from DEMO is considered a conceptual process fragment. While the procedure does not describe how a mockup should look like

on the lowest level, by means of example it has been shown this gap is easily bridged. As at the time of performing this exploratory case study, there was not a tool available that support the automatic generation of mockups. Although a counterpart implementation to support the procedure has not been created, the procedure can still be considered of the kind conceptual-implementable.

7.4.2 Implications For Future Research

While generating mockups to validate and extend DEMO models has proven to be valuable, the actual working software still has to be produced. Prescribing software architecture principles to support the differences in both enterprise ontology and implementation is easier than implementing those in software. In order to further validate that it is possible to create actual software based on these input models, follow-up research should focus on actual implementation in software, specifically addressing the level of flexibility in enterprise implementation.

Parts of this chapter are originally published as M. R. Krouwel and M. Op 't Land, *Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems* [262].[1]

8

ECS 3: Deriving a Normalized System

Abstract. To effectively respond to environmental changes, such as in market needs, technology, regulations or law, enterprises need to be able to change their supporting software accordingly. DEMO has shown to be an effective tool in designing and realizing agile enterprises. The Normalized Systems approach, on the other hand, has proven to be the key for developing adaptable software, to support agile enterprises. It was found that DEMO and its underlying theories, and the theorems and elements of the NS approach match. Moreover, an automatable algorithm was defined that allows to derive a Normalized System from the ontological model of an enterprise, while retaining some freedom in the implementation of the enterprise. This algorithm was evaluated on two real-world cases of a Dutch governmental agency responsible for the granting of subsidies. By deriving a Normalized System from DEMO, the impact of changing an enterprise implementation decision on the software can be clearly determined, and, due to the characteristics of a Normalized System this impact is minimal. With this result, one cornerstone to support enterprise agility has been covered.

8.1 Problem Formulation

To effectively respond to environmental changes, such as in market needs, technology, regulations or law, enterprises need to be able to change their supporting software accordingly. DEMO has shown to be an effective tool in realizing agile enterprises [124]. The Normalized Systems approach has proven to be the key for

[1] A detailed mapping from the original paper to this thesis can be found on p. xxx ff.

developing adaptable software [298]. Both DEMO and NS are aimed at finding concepts to support enterprise agility, comprised in both a way of thinking and a way of modeling. This exploratory case study explores how concepts from both theories can be connected in order to support agile enterprises with adaptable software.

In the Netherlands, several subsidy agencies, responsible for even more subsidy schemes, exist. While the process of granting a subsidy is rather similar for all schemes, the (business) rules differ per schema. Moreover, there are big differences in how these schemes are supported: some are more automated while others require more manual decisions. These subsidy schemes exist in a landscape with changing rules and regulations, including the obligation for the agencies to offer online channels to the interested parties to request such subsidies. It is thus desirable to have software that can easily be adapted to changing rules, regulations and enterprise implementation. Similar interests are found at the Dutch Ministry of Defense (MoD) [343]. Therefore, it was decided to explore the derivation of a Normalized System from the ontological model of an enterprise, while supporting changes in rules, regulations, and enterprise implementation.

This exploratory case study reports on an automatable algorithm to derive a Normalized System from the ontological model of an enterprise, while retaining some freedom in the implementation of the enterprise. The chapter is structured as follows: the theoretical background for this chapter can be found in Section 4.2 and Section 3.4.3; in Section 8.2 the similarities and differences between the theoretical backgrounds of DEMO and Normalized Systems are explored, the designed algorithm is shown, and the evaluation of the algorithm on the real-world case of subsidy agencies is elaborated; in Section 8.3 the results and learnings of this exploratory case study are summarized, and in Section 8.4 the learnings are formulated in terms of the goals within this research.

8.2 Building, Intervention and Evaluation

First, a high-level comparison of the theories behind DEMO and Normalized Systems are provided, after which the algorithm to create a Normalized System from a DEMO model is elaborated. Then the real-world case of Dutch governmental subsidy schemes is outlined and the required software adaptability is formulated, after which the specific output of applying the algorithm to the input models is shown, followed by a reasoning on the achieved adaptability on the software level.

8.2.1 Theoretical Comparison

For Normalized Systems it holds that, while the elements completely enforce adherence to theorems 2, 3, and 4, it is up to the designer to fully adhere to theorem 1 (separation of concerns): "the more fine-grained the identification of the tasks by a designer, the more tasks are separated from each other" [297, p. 112]. Normalized Systems identifies two different kinds of tasks: functional tasks (task elements) and generic supporting tasks (cross-cutting concerns).

The CTP (see Section 4.2.1) helps in identifying the different functional tasks, discerning two different types of tasks, namely, *a)* tasks regarding coordination, and *b)* tasks regarding production. The CTP defines the coordination and production acts that are the same for every transaction. Finally, DEMO discerns tasks in different layers, namely, original, informational or documental (see Section 4.2.3). The question then rises how DEMO helps in identifying the generic supporting tasks or cross-cutting concerns.

Typical NS cross-cutting concerns include persistency for data elements, logging for task elements and authorization (for both). Persistency typically follows from the fact that the data needs to be remembered so that it can be shared later on. It seems that the informational and documental layers may be the source of a selection of the generic supporting tasks. Logging who did what, in a way is explicitly defined in the DEMO AM, as it needs to be checked in the assess part of every ARS. This logging is tightly connected with authorization that needs to make sure that only persons who are authorized to perform a certain task can indeed do so. While the ontological model of an enterprise is abstracted from people, and thus from authorization, it does provide a solid base in terms of actor roles to authorize persons to. In a sense, the need for these cross-cutting concerns can be derived from the DEMO models.

8.2.2 Algorithm

The steps to create a Normalized System from a DEMO model are described below. The mapping from DEMO concepts to Normalized Systems concepts is summarized in Table 8.1.

1. For each entity type and product kind in the DEMO FM, a Normalized Systems data element is created. Property types become links between the data elements. Every attribute type of some entity type becomes an attribute in the respective NS data element. Value types must be represented by a software primitive.

   For the DEMO existence and occurrence laws, it is not immediately clear how they can be supported by (an) NS element(s). Additional tasks or other means of constraints might be needed. Derived fact types can be implemented in an NS task element.

2. For each transaction kind in the CM, two data elements are created: one for the initiator and one for the executor, both having their own separate flow element with task elements (see Figure 8.1). Both data elements are linked to the relevant data elements that represent the entity types in the product kind of the transaction kind. If the initiator or the executor of a transaction kind is outside the current scope, the elements for that part can be left out. The reason for having two separate workflows instead of just one, is to support the easy reorganization of actor roles over different enterprises, as common in the splitting and allying of enterprises [339].

   The two workflow patterns in Figure 8.1 are subject to interaction, as follows: in the request action of the initiator, the data element for the executor

DEMO concept	NS element: Data element	Task element	Flow element	Connector element	Trigger element	Cross-cutting concern
Fact kind	X			X		
Product kind	X			X		
Existence law		?				
Occurrence law		?				
Derived fact type		X		X		
Actor role						X
Transaction kind	X		X	X		
Transaction kind step kind		X		X		
Causal link		X		X		
Wait link		X		X		
Action rule		X		X		

Table 8.1: Mapping from DEMO concepts to Normalized Systems elements

is created; in the promise and declare action of the executor, the initiator is informed; in the accept action of the initiator, the executor is informed. To accommodate that one actor waits for the other, wait tasks are introduced. Since the initiator does not know whether the executor is going to promise or decline, it will wait for either message. Similarly, the executor waits for either an accept or reject by the initiator. Huysmans has shown how to extend these workflows to accommodate the (full) discussion and revocation patterns [219].

As actor roles will be fulfilled by subjects in an organization, the cross-cutting concern authorization should make sure that a person only can access the data and perform the tasks he or she is authorized for. The automatically generated connector elements for every data and task element are enough to support the access to fact banks.

3. The PM prescribes the exact dependencies between transaction kinds. For causal relations between transaction kinds, an NS bridge task is created. For wait links in the PM, additional wait tasks have to be introduced.

4. While most aspects of the CM have been covered by now, the decision rules itself – the assess part of the ARSs – should be placed in a separate task, to support changing the assess part while retaining the response part [468].

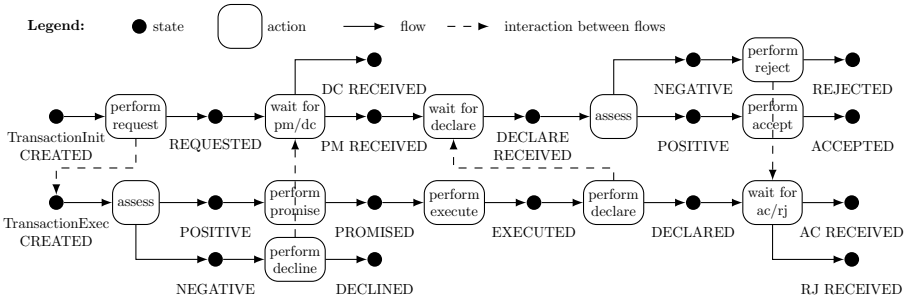


Figure 8.1: Normalized Systems basic workflow patterns for a DEMO transaction kind, expressed in two (related) State Transition Diagrams

8.2.3 Evaluation: Dutch Governmental Subsidy Schemes

Dutch law defines subsidy as the entitlement to financial resources provided by an administrative authority for the purposes of specifically named activities of the applicant, other than by way of payment for goods or services supplied to the administrative authority [105, art. 4:21.1]. In the Netherlands, many subsidy schemes exist. Moreover, these different schemes have to be supported by several subsidy agencies, while the rules and regulations, as well as the degree of automation, changes regularly. In order to support these changes, it is desirable to have software that can easily be adapted to changing rules, regulations and enterprise implementation.

Similar to ECS 1 (Chapter 7), the process of granting a subsidy is similar for all schemes. The common kernel, that was created by examining several specific subsidy schemes, including one for car demolition (Dutch: 'sloopregeling'), is shown below. In order to easily support different subsidy schemes based on this common kernel, specific requirements for the flexibility of the software are defined. For this common kernel, the output of the algorithm, i.e., the NS elements, is shown, along with a reflection on the achieved flexibility.

Enterprise Ontological Model

While different subsidy schemes exist for different kinds of activities in the Netherlands, the process of granting a subsidy is the same for all schemes (see Table 8.2 and Figure 8.2): the applicant (CTAR01) applies for a subsidy (TK01) at the subsidy granter (AR01). Before the subsidy can be granted, an evaluation has to be performed (TK02). Only if the evaluation yield a positive result, a subsidy can be granted. Next, after this evaluation, the amount the applicant will receive for his activities is determined (TK03). There can be cases where the amount is set to 0 (zero). Both the subsidy evaluator (AR02) and amount determiner (AR03) need to have access to the particular subsidy scheme act in order to know the specific criteria and the rules for amount determination. Dutch law states that payment (TK04) has to follow within four weeks after the grant.

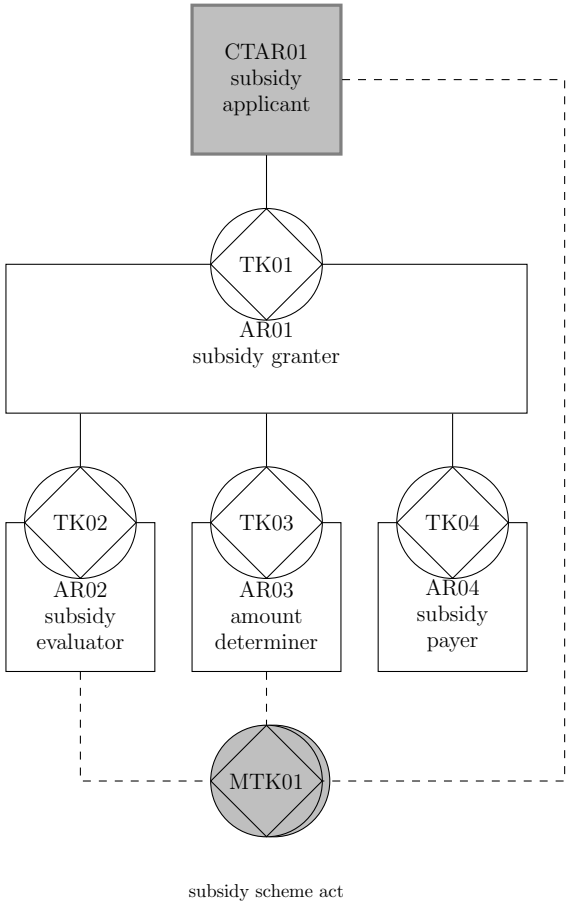


Figure 8.2: CSD for subsidy granting

Transaction Kind	**Product Kind**	**Executor Role**
TK01 grant subsidy	PK01 [subsidy] **is** granted	AR01 subsidy granter
TK02 subsidy evaluation	PK02 [subsidy] **is** evaluated	AR02 subsidy evaluator
TK03 amount determination	PK03 amount for [subsidy] **is** determined	AR03 amount determiner
TK04 subsidy payment	PK04 amount for [subsidy] **is** paid out	AR04 subsidy payer

Table 8.2: TPT for subsidy granting

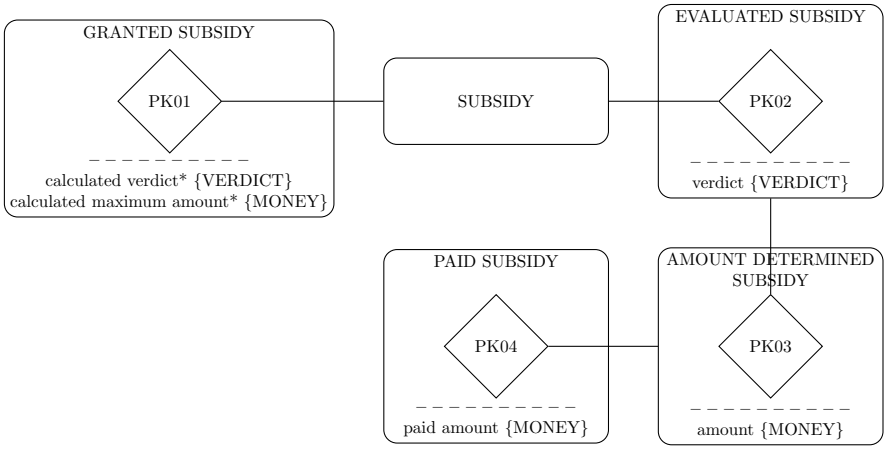


Figure 8.3: OFD for subsidy granting

From the FM (Figure 8.3) it can be read a subsidy has a derivation rule for whether it should pass the evaluation or not, and one for calculating the (maximum) subsidy amount. The evaluation result is modeled as a value type, with possible values 'positive' and 'negative'. The definitions of these derivation rules (DFSs) differ per subsidy scheme and are thus not provided on this level.

The ARSs for AR01 (subsidy granter) are provided in Appendix E; they are relatively straightforward as they don't contain actual decision rules in the assess part as the decision rules are specific for a subsidy scheme.

Desired Flexibility

For the process of subsidy granting, requirements have been formulated that within the software that supports this process, it must be possible to easily:

a) add rules for different subsidy schemes;

b) change rules following from a change in law for either the entire process or a specific scheme;

c) add different communication channels (following from the Dutch Reference Architecture NORA[2]);

d) choose to automate tasks or perform them manually;

e) switch between internal and external data usage;

f) choose which information is shown on a screen;

g) outsource or insource parts of the system; and

h) add controls for managing throughput time.

[2] https://www.noraonline.nl/

Output Elements

1. Five data elements are created: Subsidy, GrantedSubsidy, EvaluatedSubsidy, AmountDeterminedSubsidy, and PaidSubsidy, Money (see Figure 8.4). Money is represented by a floating point number, and Verdict is represented by a string – an alternative is boolean, but the NS approach states that booleans should be avoided as they lock up the possibility to have more than two values. While it might be possible to define generic tasks to calculate the derived facts, the implementation will be specific for a scheme and, as a result, these tasks are not defined at the generic level.

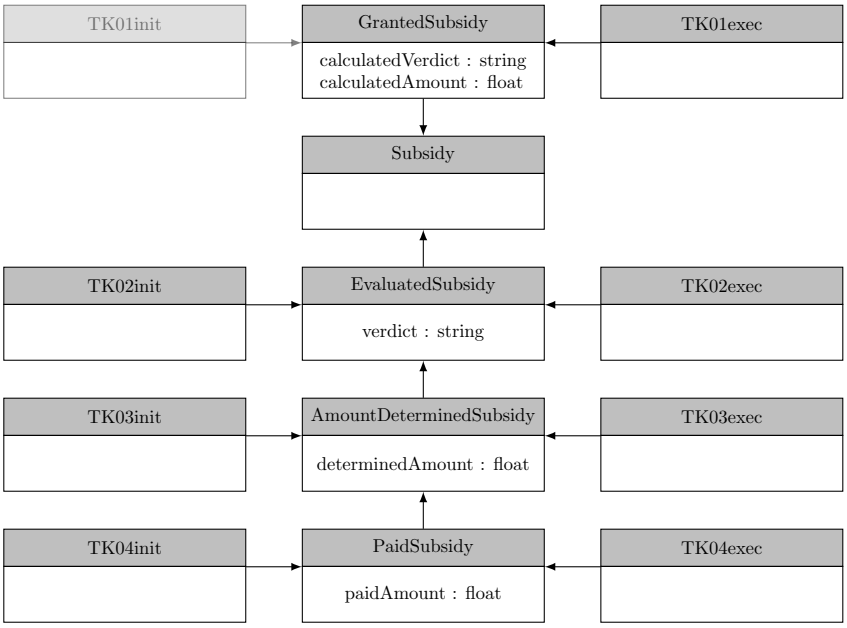


Figure 8.4: ERD of identified NS data elements for subsidy granting

2. As three of the four transaction kinds reside within the focus, $3 * 2 + 1 = 7$ data elements are created to support the workflows (see Figure 8.4). Additionally, a data element for TK01/rq could be created in case the software needs to support the performing of a subsidy grant request (shown grayed out in Figure 8.4). Every data element is linked to the data elements representing its product kind.

3. Identified bridge tasks follow from the fact that TK02, TK03, and TK04 are enclosed in TK01: *create TK02init*, *create TK03init*, and *create TK04init*. Relevant wait tasks follow from the wait links in the PM, also shown in the while clauses in the ARSs: *wait for TK02/ac*, *wait for TK03/ac*, and *wait for TK04/ac*. The complete workflow for T01exec is shown in Figure 8.5. The other flows don't deviate from the basic workflow as shown in Figure 8.1.

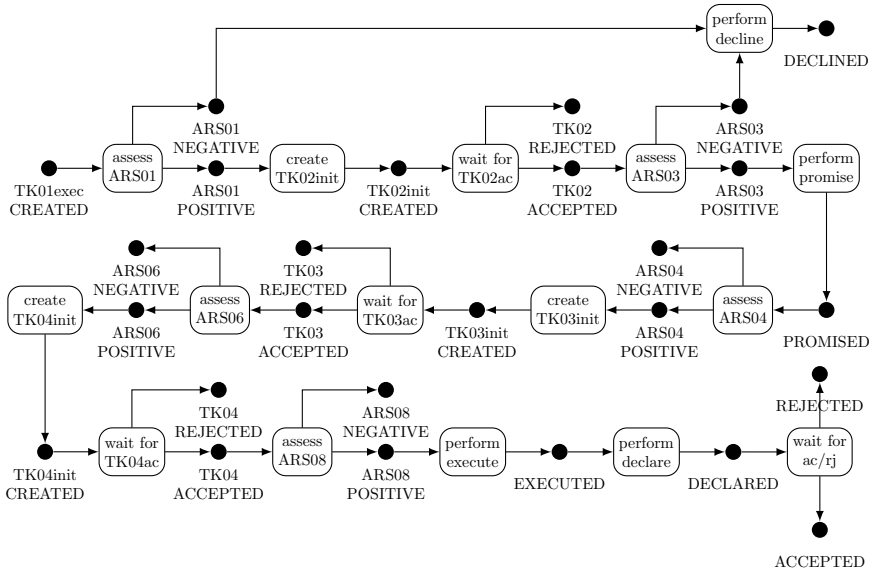


Figure 8.5: Workflow for the T01exec data element

4. While the basic pattern already covers the assessment of ARS01, ARS02, ARS05, and ARS07, additional tasks to evaluate the assess parts (decision rules) of ARS03, ARS04, ARS06, and ARS08 (see Figure 8.5) are created.

Achieved Flexibility

By applying the algorithm, a set of NS elements is generated, that, when put into the descriptor files, can be expanded automatically using the NS expander (see Section 3.4.3). Below, a reasoning will follow to what extent the desired flexibility is achieved.

a) Add rules for different subsidy schemes: Because of data version transparency, different implementations of the subsidy data element for different schemes can easily be introduced. For example, for one scheme, the subsidy is about cars, while for another scheme, it is about houses. Because of action version transparency, different actions that apply for the different kinds of subsidy can be implemented as well. For different schemes, the task for calculating the maximum amount will differ. Of course, there must be some mechanism that ensures the data and task versions are matched.

Alternatively, it could be argued to generate completely new (and separate) software for each subsidy scheme, based on the generic set of elements.

b) Change rules for the entire process or a specific scheme: This is similar to adding a new scheme. Depending on the nature of the change, this will

mean implementing a new version of some task or adding or deleting some task(s).

c) Add different communication channels: Because of task version transparency, different versions of a task to perform a C-act can be implemented. In general, to enable reuse, these tasks might become bridge tasks with their own designated workflow.

d) Automate tasks or not: As before, this is up to the implementation of a task. If it is chosen to automate the task, it must be implemented in the software. Otherwise, a manual task is used. By means of task version transparency, several implementations (versions) can coexist.

e) Switch between internal and external data usage: For example, for getting person data, e.g., about the applicant of a subsidy (currently not modeled), the enterprise could use its own database in which it must enter all person data on first use. However, it could also be decided to connect to some governmental person register. Because of data version transparency, it is easy to change this choice.

f) Decide which information is shown on the screen: A screen is user interaction that is provided by connector elements. Creating a new or changed screen means adding or changing a connector element. This has no impact on the task or data elements.

g) Outsource (or insource) parts of the system: A similar reasoning holds as for changing rules for the entire process: either tasks are deleted or they are re-implemented as external task. By splitting the workflow of the initiator and executor (see Figure 8.1), the flexibility in the resulting software does not only help to support the splitting of enterprises, but also eases allying with new partners [339, p. 102].

h) Add controls for managing throughput time: This can be reached by adding trigger elements with timers. For example, if the payment of some application is not performed within three weeks after granting, some notify task must be started.

8.3 Reflection and Learning

In linking DEMO and Normalized Systems theory, it was found that the underlying theories both prescribe a modular structure and share the idea that separation of tasks is crucial to fully design an (agile) system. The ontological model of the O-organization of an enterprise provides enough information – assuming some mapping from value types to software primitives – to create fully functional software that supports the operation of the enterprise and provides freedom in implementation. This implementation freedom includes, but is not limited to, using different information channels, choosing to automate tasks or not, and changing business rules, and therefore is considered a cornerstone to support enterprise

agility. Similar results, including a big gain in the development speed, were found at the MoD [343]. .

At the same time, given the choice for DEMO and Normalized Systems, the software is not optimized for several non-functional requirements such as user-friendliness and performance, as these aspects are not specified in a DEMO model. The Normalized Systems approach considers these requirements as cross-cutting concerns that should be addressed separately from the functional requirements. These aspects should therefore be added to the input models, in order to create fully compliant software.

Unfortunately, from this exploratory case study and additional evaluations at the MoD, it is not clear for all DEMO concepts how they can be supported by Normalized Systems elements. As can be read from Table 8.1, for the existence and occurrence laws it is not yet clear how to model them in a Normalized System; there is not a construct in NS onto which it can directly be mapped.

The other way around, from Table 8.1 it can also be read for each NS element which DEMO concepts are needed in order to define it. The empty column of trigger element shows that there are no immediate clues in DEMO models that provide the need to define a trigger element.

8.4 Formalization of Learnings

In this exploratory case study, the comparison of the theories behind DEMO and Normalized Systems showed that they are highly compatible. As both are helpful in designing adaptable systems, the combination should provide a solid base to support enterprise agility. An algorithm was created to derive a Normalized System from the ontological model of an enterprise, with specifically designed adaptability in terms of both enterprise ontology and enterprise implementation.

While this exploratory case study was performed prior to the existence of the EIF (Section 4.3), multiple OIVs are supported implicitly. As the system can be regenerated for new products and services, the algorithm thus supports all three kinds of change as defined in need 2. This again underpins the idea that it is possible to create software that has a stable core, based on DEMO, and is adaptable around that core, as defined in chosen OIVs.

8.4.1 Identified Fragments

The conceptual mapping from the DEMO metamodel to the NS metamodel is considered a conceptual-implementable process fragment. The conversion from DEMO models to NS descriptor files has been performed manually, and no technical counterpart fragment has been produced.

8.4.2 Implications For Future Research

This is the first exploratory case study where working software has been produced. At the same time, there are still manual conversions in the process, introducing the possibility for errors and/or implicit design decisions. Moreover, while some

dimensions for enterprise implementation flexibility were defined, this exploratory case study did not build upon the EIF. When the software can easily, thus automatically, be regenerated with changed input models, the question also arises whether the software strictly has to adhere to all the NS theorems. Follow-up research should focus on the automatic conversion from DEMO to working software, accommodating explicitly designed enterprise implementation flexibility, based on the EIF, so that it can be checked whether the NS expanders are actually a necessity to construct adaptable software.

Parts of this chapter are originally published as M. R. Krouwel, M. Op 't Land, and H. A. Proper, *Generating Low-Code Applications from Enterprise Ontology* [266].[1]

9 ECS 4: Generating Mendix Applications

Abstract. Due to factors such as hyper-competition, increasing expectations from customers, regulatory changes, and technological advancements, the conditions in which enterprises need to thrive become increasingly turbulent. As a result, enterprise agility becomes an increasingly important determinant for enterprise success. Since software development often is a limiting factor in achieving enterprise agility, enterprise agility and software adaptability cannot be viewed separately and choices that regard agility should not be left to developers. By taking an MDSD approach, starting from ontological models of the enterprise and explicit design decisions, the gap from enterprise agility to software adaptability is bridged, in such a way that changing software is no longer the limiting factor in changing the enterprise. Low-code technology is a growing market trend. In this exploratory case study, a mapping is created for (the automation of) the creation of a Mendix low-code application from the ontological model of an enterprise, while accommodating the explicitly defined enterprise agility. Even though the algorithm has been evaluated successfully on multiple cases, supporting all possible enterprise implementation variability seems to be an $\mathcal{NP}$-hard problem, and more research is required to check the feasibility and usability of this approach.

9.1 Problem Formulation

Currently, enterprises are mostly adaptive at design-time, where transformation projects and programs that change the operation can take between a few weeks

[1] A detailed mapping from the original paper to this thesis can be found on p. xxx ff.

and several years. Run-time adaptive enterprises are enterprises with a near-zero time-to-market for new or changed products and services, implying that the software time-to-market is not on the critical path of its business time-to-market [342]. In this exploratory case study, the hypothesis is that, by taking an MDSD approach towards low code, starting from enterprise ontology, the support of enterprise agility can be improved up to the point that software development is not the limiting factor anymore.

As introduced in Section 4.3, consciously and explicitly designing enterprise implementation is necessary in order to support enterprise agility – otherwise such design decisions may end up hard coded in software and thus hard to change. Existing method fragments seem to ignore the concept of Enterprise Implementation completely (see Section 5.2). A potential explanation for the latter is that creating code to support a mapping of the complete metamodel of a modeling method such as DEMO is a highly complex and time-consuming task. This is also why, in principle, the researcher suggests using a MDSD or code generation strategy, as real-time interpreters can become even more complex. At the same time, such an approach can make it harder to make (controlled) customizations and extensions that are often needed in practice. This is why the researcher suggests turning to low-code technology as it allows for controlled insertion of custom code.

This exploratory case study reports on the design of an automated model transformation to create a Mendix[2] low-code application from Enterprise Ontology (expressed as DEMO models) and Enterprise Implementation (expressed in terms of OIVs). This chapter is structured as follows: the relevant background for this chapter can be found in Section 4.2 (Enterprise Ontology), Section 3.3.2 (MDSD), Section 4.3 (Enterprise Implementation), and Section 3.4.4 (low code in general); in Section 9.2 the choice for Mendix is discussed, and the mapping including its validation and evaluation in practice is shown; in Section 9.3 the results and learnings of this exploratory case study are summarized and in Section 9.4 the learnings are formalized in terms of the goals of this research.

9.2 Building, Intervention and Evaluation

The Mendix low-code platform is chosen because it provides good documentation about its metamodel,[3] that is a specialization of the more generic low-code metamodel (see Table 9.1), and offers a Software Development Kit (SDK) to create Mendix applications using TypeScript.[4] Next to that, the research is experienced in using the Mendix platform in real-world situations.

In order to be able to (automatically) generate Mendix applications from DEMO models, a (conceptual) mapping from the DEMO metamodel (see Figure 4.7) to the Mendix metamodel is devised. Initially, this mapping was done manually, while later a Typescript (reference) implementation of this mapping[5]

[2] https://mendix.com/

[3] https://docs.mendix.com/apidocs-mxsdk/mxsdk/mendix-metamodel/

[4] TypeScript is the language to access the Mendix SDK, also known as model API, see https://docs.mendix.com/apidocs-mxsdk/mxsdk/.

[5] Source code is available at https://github.com/mkrouwel/demo2mendix.

Low-Code Unit	Mendix Unit
Data	Entity with Attributes and Associations, Enumeration
Logic	Microflow
Screen	Page, typically containing Action Buttons
Permission Rule	Access Rule

Table 9.1: Mendix (metamodel) units as specialization of generic low-code metamodel (Figure 3.8)

was created to allow for easy validation with multiple DEMO models. Both the mapping and the implementation have been evaluated several times with different input models, ranging from academic cases such as EU-Rent [334, 341] and Volley [124] to real-world cases including Social Housing [342]. While the evaluations with different cases improved and enhanced the mapping and its implementation in order to deal with all the required concepts and ease the process of code generation, the evaluations never showed the need for a major redesign of the mapping or its implementation.

The final mapping includes concepts from both the DEMO metamodel and the concept of OIV and can be found in Table 9.2. The result of applying the mapping function to a DEMO model is a full-functional Mendix low-code application containing a data model (and database), (basic) CRUD screens, a security model so that users can only access the parts they are authorized for, several workflows to support the DEMO CTP, and (business) rules for the evaluation of DFSs and truth parts of the ARSs. It is fairly easy to add APIs on top of the data model and/or logic to expose data or functions to other applications. In the process of creating the mapping, several design decisions have been made that are reported below.

D.1 Transaction Kinds are not mapped. Instead, their associated Product Kinds are mapped to an Entity in order to be able to capture the state of a transaction, see D.4.

D.2 For Multiple Transaction Kinds (MTKs) it is usually not needed to capture the coordination acts around these facts, so no mapping is needed. The production facts in the MTK are present in the FM and mapped accordingly to a Mendix unit, see D.5.

D.3 As the page for showing the agenda for an actor is a very generic functionality, it was not generated but built in Mendix as a reusable component (module). The logic to support the state machine representing the Complete Transaction Pattern is also built as a generic module. The details of this module are not part of this chapter as they merely rely on earlier research (see Section 5.2). Development of this module has started before Mendix launched its native workflow capabilities, that has the potential to reduce complexity of this module.

DEMO Concept (Aspect Model)	Example	Mendix Unit
Elementary Transaction Kind (CM)	TK01	n/a, see D.1
Multiple Transaction Kind (CM)	MTK01	n/a, see D.2
Actor Role (both elementary and composite) (CM)	AR01	User Role, see D.3
Executor Link (CM)	AR01-TK01	Action Button and (Microflow) Access Rule
Initiator Link (CM)	CAR01-TK01	Action Button and (Microflow) Access Rule
Access Link (CM)	CA01-MTK01	Entity Access Rule
Product Kind (CM)	[registration] is started	Entity having *Transaction.Proposition* as generalization, with Association(s) to its variable(s), see D.4
Transaction Kind Step Kind (PM)	TK01/rq	Page
Declared Entity Type (FM)	Registration	Entity and Pages, see D.5
Aggregation Entity Type (FM)	{Registration X Year}	Entity with Associations to its aggregates
Specialization Entity Type (FM)	Started Registration	n/a, see D.6
Generalization Entity Type (FM)	n/a	n/a, see D.6
Property Type (FM)	member	Association
Declared Attribute Type (FM)	starting day	Attribute or Association, see D.7
Calculated Attribute Type (FM)	age	Microflow, see D.8
Value Type (FM)	day, money	Enumeration or Entity, see D.7
Action Rule-Event Part (AM)		Action Button and (Microflow) Access Rule, see D.9
Action Rule-Assess Part (AM)		Microflow and Page, see D.9
Action Rule-Response Part (AM)		Action Button, Microflow and (Microflow) Access Rule, see D.9
Organization Implementation Variable	Functionary Type	Entity, see D.10

Table 9.2: Mapping from the DEMO metamodel to the Mendix metamodel

D.4 *Transaction.Proposition* is an Entity that is part of the generic module handling the CTP. By extending it through specialization, the generic state machine can be used, while it is still possible to relate the specific product entity to the entity or entities the Product Kind is about (the variables in the product formulation).

D.5 For out-of-focus Entity Types in the FM, the decision has to be made whether the data is stored within the generated application, or used from another source, typically through an API. For the latter, in Mendix an external entity can be created, but it requires the API to be available in Mendix Connect.[6] For now it was decided to not use that possibility, especially as this does not seem possible (yet) through the Mendix SDK. Instead, some basic CRUD (Create, Read, Update, and Delete) pages are created to modify and view the data. It is fairly easy in the generated application to change this later.

D.6 There were not enough example DEMO models to provide a mapping for the generalization and specialization Entity Types.

D.7 DEMO Attribute Types can have different kinds of Value Types. If the scale sort of a Value Type is categorical, the Value Type can either be mapped to a Mendix Enumeration or Entity. The DEMO Attribute Type using the Value Type will then either be a Mendix EnumerationTypeAttribute or Association. If the scale sort of the Value Type is of some other type, e.g., day, money, etc., the DEMO Attribute Type will be mapped to some specific Mendix AttributeType, e.g., DateTime, Decimal, etc.

D.8 Calculated Attribute Types need to be calculated, for which currently a Microflow is created. The (mathematical) definition of the calculation (as defined in a DFS) needs to be implemented in the Microflow, of which the mapping is too extensive and detailed for the scope of this chapter. A decision that goes along with this choice is that from a performance perspective there is still the decision whether the calculation is performed on request (read) or on change (save). The low-code approach makes it easy to make such a decision in the platform, as it currently seems too difficult to include this aspect into the mapping.

D.9 For the handling of Action Rules, it was decided to implement this as a Mendix Action Button for the User Role that has to deal with the agendum (type), a Mendix Page to see all the relevant information to decide on a response, as well as one or more Mendix Action Buttons for the different choices. In this way the autonomy of the actor(s) involved is respected, and only the parts for retrieving, and possibly calculating, all the information are automated. The transformation of the assess-part is similar to that of a

[6] Mendix Connect is a collection of functionalitities that enables developers to discover and use APIs from other applications and publish APIs for other applications to use, see `https://www.mendix.com/data-hub/`.

Calculated Attribute Type and thus a similar reasoning holds as described in D.8.

D.10 In this research the principle is adopted that a chosen set of OIVs should be adaptable at run-time, and therefore this concept is mapped to a Mendix Entity, including basic CRUD pages to edit the value, i.e., design decision, of a certain OIV in the running application. The different values of such an OIV can have impact on authorization, redirecting and handling of a C-act, and much more. This is considered to be part of the logic and state machine and supports the choice for low code as target platform because it is easier to build this into the state machine than into the automated translator. At the same time, in Section 4.3.4 it has been reasoned that the possible number of configurations can grow exponentially with the number of OIVs (see Section 4.3); incorporating OIVs into the logic might turn out to be a $\mathcal{NP}$-hard problem.

The Social Housing case is used as an illustration.[7] Within the domain of Social Housing it is important to be able to easily shift responsibilities between organizational units, e.g., from one social housing association to an umbrella of social housing associations, or to a municipality, or to a collective of municipalities. Moreover, the assignment of actor roles to functionary types shifts regularly, due to changes in required education and competence level, compliance requirements and labor market opportunities or constraints. Finally, as it is common practice to authorize persons to fulfill functionary types, and not actor roles directly, it was decided to select *Organizational Unit*, *Functionary type* and *Authorization* from the collection of OIVs (see Section 4.3) as the OIVs that should be easily changeable in the supporting software.

Figure 9.1 shows the descriptor file for Social Housing that was fed into the automated translator to generate a Mendix application. The descriptor file includes an app name and module name and refers to the DEMO model in JSON format (see Appendix C). Figure 9.2 shows the project folder of the generated Mendix application, containing pages with buttons, microflows and enumerations. Figure 9.3 shows the generated domain (data) model consisting of entities, attributes and associations. Additionally, the needed user roles are created and all Mendix units have the proper access rules so that only authorized users can see and/or mutate the data and/or use the functionality provided through screens. Figure 9.4 shows how the truth part of an action rule was implemented in Mendix and Figure 9.5 shows a screenshot of the running application where an actor with the right authorization can deal with a C-act of kind TK01/rq.

```
{"appname": "SH", "defaultmodule": "SH", "demomodel": "./SHdemomodel.json"}
```

Figure 9.1: Descriptor file in JSON format for Social Housing

[7] For the case description see Chapter 6.

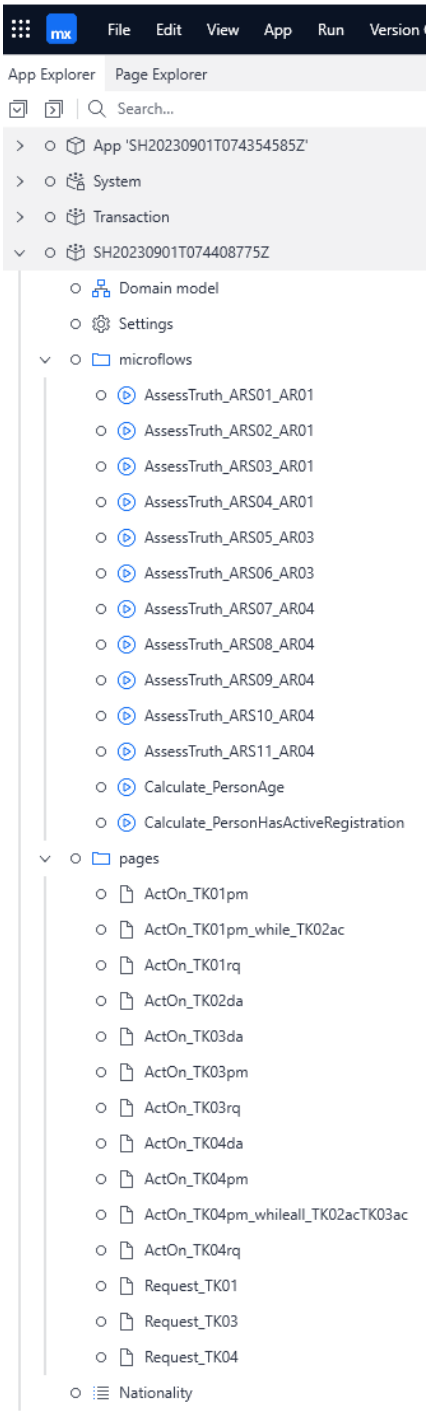


Figure 9.2: Project outline of the generated Mendix application for Social Housing

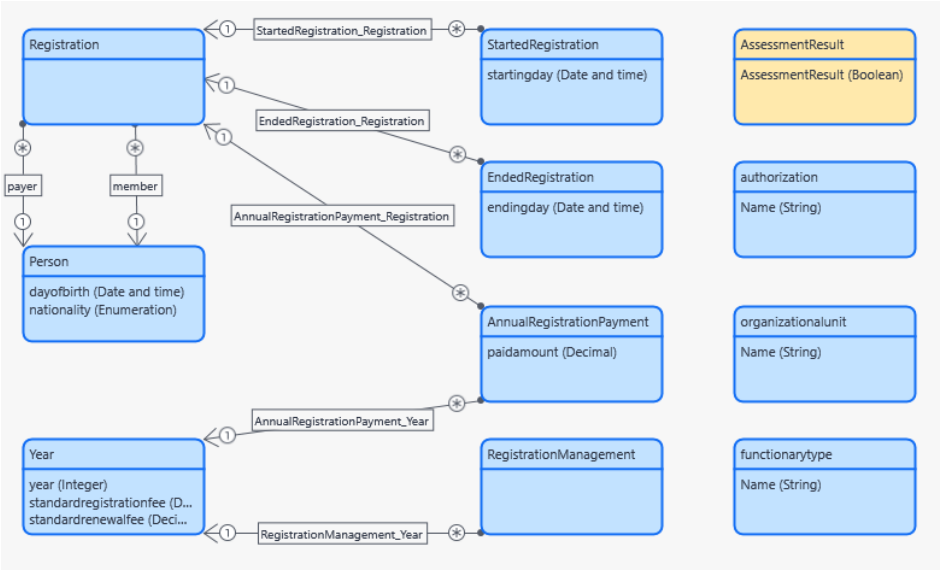


Figure 9.3: Domain (data) model of the generated Mendix application for Social Housing

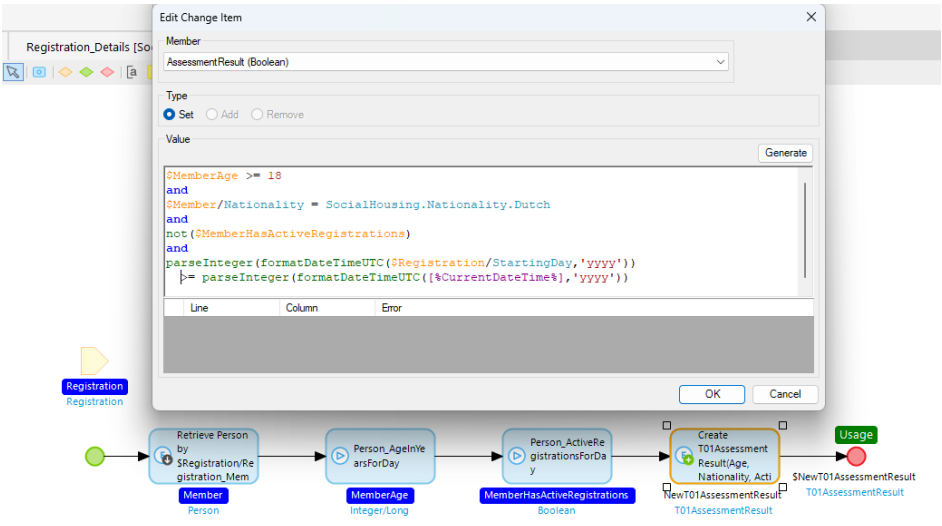


Figure 9.4: Mendix implementation of the 'truth part' of ARS01

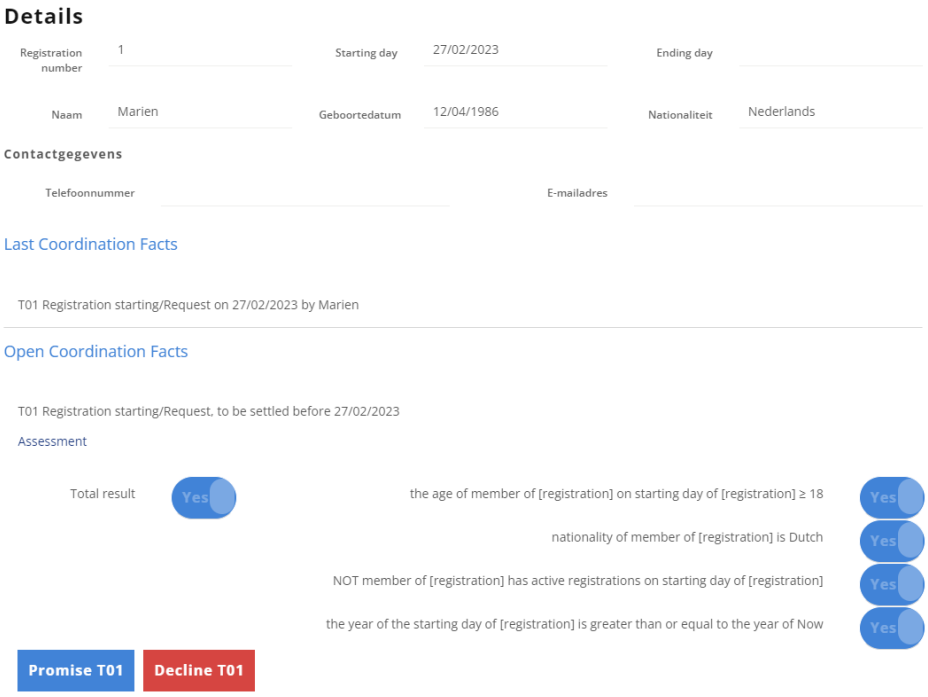


Figure 9.5: Example screen for AR01 to deal with TK01/rq

9.3 Reflection and Learning

By introducing the concept of Organization Implementation Variable into the MDSD process or code generation, it became able to create transparency in the mapping from enterprise design decision(s) to software implementation, thereby not leaving it up to developers to make the right choice. As the enterprise's implementation is configurable in terms of OIVs, it is easy to change the enterprise implementation (or: configuration) for a given enterprise's ontological model, thereby supporting enterprise agility.

Given the choice for Mendix as low-code target platform, a complete mapping from the DEMO metamodel including enterprise implementation to the Mendix metamodel was defined. This mapping was implemented in TypeScript, using the Mendix SDK, in order to generate a readily deployable low-code application from the ontological model of an enterprise, including the support for run-time configurable OIVs. During development of the reference implementation, reusable components in Mendix were created to support the CTP as well as for showing relevant agenda to the actors that have to deal with them.

One advantage of the proposed approach follows from the use of an ontological enterprise model as the starting point. Since these models are comprehensive, coherent, consistent, and concise (see Section 4.2), the generated software is of high quality and only contains the necessary constructs to support the enterprise's

end users. It also adds flexibility as parts of the software system are simply (re)generated when new transaction kinds arise. By adding OIVs to the MDSD approach, it becomes possible to change (some) design decisions at run-time, therefore allowing for even more flexibility and agility for the enterprise.

Another advantage of this approach seems to be that the generated code can easily be adapted through the low-code visual paradigm. This allows for changes in the UI, to make use of APIs, or to implement the execution of the action rules or calculations in a more efficient way. A warning should be given that changes in the generated code can become a source of hidden design decisions.

Reflecting on the MDSD approach, by making explicit the required enterprise flexibility and giving it a specific place in the generated code, the notions of enterprise flexibility and software flexibility have been connected. As the code can easily be regenerated, changing the software system is not the limiting factor anymore in changing the enterprises service or product portfolio or its implementation.

The implementation of OIVs in Mendix, or maybe even software in general, is not straightforward. The impact of certain software design choices in the state machine and logic is not completely detailed yet. This will differ per OIV and can only be detailed per individual OIV. It can even be that when combining different OIVs, the problem becomes to complex to solve. It is not clear whether all OIVs can be implemented completely independent of others, as suggested by Normalized Systems theory (see Section 3.4.3). More evaluations with more cases involving more OIVs are needed to fully understand the complexity of this problem. Supporting all possible enterprise implementations could even be an $\mathcal{NP}$-hard problem. Further research is required to check the feasibility and limitations of this approach.

The mapping is specific towards the Mendix platform. Although other low-code platforms rely on similar concepts, the question arises whether the mapping can be abstracted to facilitate other low-code platforms, or even high code. Further research is needed to get a perspective on the feasibility and usability of such an abstraction.

Mendix and other low-code platforms offer ways to easily create and consume APIs. The current mapping does not build on the findings of the exploratory case study on microservices (Chapter 6). It could be interesting to see how these two can be combined so that not only generate the API definitions are generated but also their implementations.

9.4 Formalization of Learnings

In this exploratory case study, the hypothesis was that by taking an MDSD approach towards low code, starting from enterprise ontology, the support of enterprise agility can be improved up to the point that software development is not the limiting factor anymore. By starting from an enterprise's ontological model and introducing the concept of OIV into the MDSD process, it was shown that it is possible to automatically generate a (Mendix low-code) application that supports

changing an enterprise implementation at run-time. As changing an enterprise's implementation often has more implications than just changing the software, at least there now is a possibility to guarantee that software development is no longer on the critical path of changing an enterprise implementation and/or launching new or changed products and services.

By not only creating a (conceptual) mapping, but by also implementing the mapping, software that is adaptable at run-time to identified OIVs can be generated from DEMO models. As (parts of) the application can easily be regenerated for new products and services, all three kinds of change as defined in need 2 are in fact supported. This exploratory case study also underpins the idea that it is possible to create software that has a stable core, based on DEMO, and is adaptable around that core, as defined in the chosen OIVs.

9.4.1 Identified Fragments

The conceptual mapping from the DEMO metamodel to the Mendix metamodel is considered a conceptual-implementable process fragment. Its counterpart implementation in TypeScript is a technical process fragment.

Part III

Results

"Glory lies in the attempt to reach one's goal and not in reaching it."
Mahatma Gandhi, in 'A Young Canadian's Question' (1942)[1]

10 Conclusions

In Chapter 1 the background and motivation of this research is outlined, i.e., that, in order to support enterprise agility, enterprise software should support end users (need 1), should be adaptable (need 2), should be created quickly (need 3), and that it should be traceable how the end users' requirements are supported by the software (need 4). It is argued that the only way to provide an answer to the identified needs is by having a structured method. Such methods are not new and date back to the 1970s, but advancements in Enterprise Modeling and software technologies have paved the way for a new method to address the needs. Partial answers are found in Model-based Engineering and (enterprise) models but it is also shown there are still many uncertainties to create such a method. Therefore, it is concluded that this research is in the fuzzy front-end stage of creating such a method, aimed at better understanding the challenge and possibly providing initial solution elements. A main research challenge is formulated along with seven research questions to deal with the uncertainties in this challenge. An Action Design Research approach consisting of both literature review and exploratory case studies is adopted to answer the research questions and to be able to reflect on the main research challenge.

In this chapter the contribution to a method for software development that supports typical enterprise changes will be reviewed. In Section 10.1, answers to the research questions will be formulated, ending with a reflection on the main research challenge. In Section 10.2 the research contributions are formulated, and in Section 10.3 the broader impact of this research is outlined. A critical reflection on this research as well as its limitations and directions for future research will follow in Chapter 11.

[1] Published as writing 526 (March 26, 1942) in 'The Collected Works of Mahatma Gandhi', volume 75.

10.1 Answers to Research Questions

The main research challenge is decomposed into theoretical and more practical research questions. Initial answers to the theoretical questions are provided in Part I. In Part II the initial answers to the theoretical research questions are validated, while also trying to provide answers to the practical research questions. In this section the answers to the theoretical research questions are reviewed, followed by the answers to the practical research questions, in order to be able to reflect on the main research challenge.

10.1.1 Answers to Theoretical Research Questions

Step-wise Creation of Software

RQ 1: *What does it mean to step-wise create software for enterprises?*

Using models in software development has been common since the 1970s. Models provide a way to study a system, such as an enterprise or software, before it is actually implemented. Not using models in the software development process is considered a bad practice, especially in enterprise environments.

The General System Development Process (GSDP) is adopted as it seems to be the most generic framework for system design that is grounded in theory and applies a step-wise refinement process that relies on the use of models. The GSDP is aimed at deliberately designed systems, including enterprises and software. It says that, after having collected, possibly only a fraction of, the functional requirements by analyzing the construction of its environment, the (technical) design of a system consists of the subsequent creation of more detailed constructional models; it starts from the ontological model, a conceptual model that is completely independent of technology, and ends with the implementation model, the most detailed and technology-dependent model.

Model-based Engineering is the common name for any approach towards software engineering that uses models. It is shown that the more specific Model-driven Software Development approach has the potential to answer needs 3 and 4 and is fully compliant with the GSDP. As a result, the Enterprise Model-driven Software Development approach is adopted, an approach that relies on model transformations starting from enterprise models. The exploratory studies have shown that such an approach indeed works in practice and addresses needs 3 and 4.

Input Models

RQ 2: *What model(s) should be used as input for the method?*

Given the choice for applying an EMDSD approach that seems to address needs 3 and 4, the addressing of need 1 is still fully up to the chosen input (enterprise) model. Several perspectives exist in enterprise modeling, and combining multiple perspectives improves the overall quality and completeness of the model(s). Based

on the needs and the different perspectives that exist, several criteria to choose (an) enterprise modeling technique(s) are formulated, including:

- the chosen modeling technique should cover the perspectives Product and/or Service, Actors (or Roles), Process, Information and/or Data, and Business Rules;
- the chosen modeling technique should comprise a modeling language (WoM) and a modeling procedure (WoW), that is supported by a Way of Thinking (WoT) and tools (WoS);
- the metamodel and semantics of the chosen modeling technique are formally defined, as is required for MDSD; and
- there is a user group behind the chosen modeling technique with a large enough sample base.

Given these criteria, and considering that combining modeling techniques is possible but possibly too complex in this stage of the research, it is concluded that, despite the lack of proper tools, DEMO has the best fit for this research. A concern in using DEMO is that its abstraction level is far from the actual implementation. Therefore the Enterprise Implementation Framework is developed that allows expressing an enterprise implementation in terms of Organization Implementation Variables, related to concepts from the DEMO (meta)model. Using OIVs offers the possibility to make design decisions explicit and to make transparent how they are implemented in software, preventing software developers to implicitly hard code these decisions into the software. For this research, DEMO models and the OIVs are used as input models for the EMDSD approach, as together they seem to address need 1 and allow for *controlled variability*, addressing need 2.

The exploratory studies have shown that DEMO models alone are not enough to fully generate software that answers needs 1 and 2. By incorporating the concept of OIVs into the process, it has been shown that it is possible to fully generate software that is adaptable to the level that is required by the enterprise.

From ECS 3 (Chapter 8) it has been concluded that it could be necessary to add non-functional requirements, such as user-friendliness and performance, to the input models. At the same time, several of these non-functional requirements are nowadays often – implicitly or explicitly – already addressed by target platforms. In order not to (over)complicate this exploratory research, it is left to future research to find out if, and perhaps how, such non-functional requirements should be further incorporated into the input models.

Target Technologies

RQ 3: *What target technology or technologies should be supported by the method?*

In order to address some concerns of the chosen MDSD approach, i.e., *a)* that it lacks integration of custom code, *b)* that the result often has a poor UI, *c)* that it is seen as quite rigid, and in order to address need 2, a range of modern software technologies as possible target outputs has been explored. Four technologies were selected:

- mockups as they are aimed at providing a good-looking UI, while they are also used to assess the correctness and completeness of the requirements and thus addresses need 1;
- a microservice architecture that aims to provide flexibility by creating lots of small components, addressing need 2;
- Normalized Systems as it aims at creating stable systems that support continuous change, addressing need 2 and, as NS can be considered an MDD implementation by itself, it also addresses needs 3 and 4; and
- low-code platforms are a big trend that address in fact both needs 2 and 3, while it also aims at offering a good UI.

Of all these technologies the metamodel is provided that is needed to create a mapping from the input (metamodel) to the target metamodel as part of the model transformation for MDSD.

The exploratory studies have shown that the chosen input models can indeed be transformed into these chosen target technologies, that sometimes themselves apply transformations to even lower-level (implementation) models or running software.

Creating a Method

RQ 4: *What does the creation of a (situational) method entail?*

A method is a step-by-step approach to perform a system development project, based on a specific way of thinking, consisting of structured activities and deliverables, preferably supported by models and tools. In creating or configuring a method, often (proven) parts of existing methods, called method fragments, are used. Method fragments can be assembled into a more generic or a more project specific situational method. Fragments can be categorized in three axes: perspective, abstraction and granularity. In order create a method, method fragments are needed first.

Such fragments can be already existing or newly created. Given the choice for DEMO models as input models, literature has been reviewed on the existence of method fragments that start from DEMO models and end in some software implementation. Although there seem to be a few usable fragments, none of these fragments use all aspect models of DEMO and most of them are conceptual fragments that don't have a technical implementation as counterpart. Moreover, most existing fragments seem to ignore the step from DEMO to implementation model and try to immediately create software from DEMO models, of which it has been shown that it is only adaptable with respect to a small set of types of enterprise change. This provided the directions for the exploratory case studies to create method fragments that cover all aspect models of DEMO include explicit enterprise implementation design decisions, and preferably have a counterpart in software, or are very close to being executable by IT.

10.1.2 Answers to Practical Research Questions

Feasibility

RQ 5: *Is it technically feasible to create software from (the selected) enterprise models?*

A famous example of creating software from enterprise models is MDA, that uses UML models to visualize and generate software. This shows that creating or, even stricter, generating software from enterprise models is possible. There are however several known issues with MDA, including poor integration of different models, lack of efficiency, limited applicability, vendor lock-in, and its complexity. Moreover, as UML is more aimed at software modeling and has limitations with regard to enterprise modeling, it is questionable whether MDA addresses all identified needs.

Given the choice for an MDSD approach from DEMO models to several software technologies, and given the practical setup of this research, the exploratory case studies have shown that it is possible to provide a mapping from the input models to the target technologies and/or an algorithm that transforms an input model into a (computer program in the) target technology. The exploratory case studies (once more) confirmed that DEMO models provide a relatively stable foundation and showed that the OIVs provide a way to consciously decide on the implementation that supports the end users' requirements as well as the required adaptability. Moreover, by creating two different (reference) implementations, it has been shown that it is possible to automate these mappings and algorithms. Thus, it is technically feasible to automatically generate software with the selected technologies from the chosen DEMO models and OIVs, and that there are multiple ways to achieve this.

Requirements

RQ 6: *What are requirements for a method to develop software from enterprise models?*

As mentioned in Section 1.2, the requirements for the desired method were not clear. The fuzzy front-end research is aimed at creating a better understanding of the challenge and the requirements for the solution. From literature it was already concluded that having a (structured) method is better than not having one. The exploratory case studies have shown that it is possible to have such a method, and that it seems to improve the results and address the identified needs. Having a method at least provides a way of repeating the procedure, so that learnings can be taken to a next case and improvements on the method can be made with every application of the method. The real usefulness of such a method can only be validated when the 'complete' method has been created.

The exploratory case studies have not been able to prove or disprove the hypothesis that the identified needs form the base requirements for the 'complete' method. The exploratory case studies however did show the need for a flexible

target technology, i.e., that the method should be adaptable to generate software with different target platforms or technologies. It will mostly be dependent on the project specific situation what target technology will be chosen. The exploratory case studies also provide insights in the detailing of the needs, and specifically for need 2 it turned out it is important to distinguish between the different types of (expected) change so that conscious decisions can be made which do and which do not need to be supported by the software; supporting all possible variability seems to be impossible at first sight.

Method Fragments

RQ 7: *Which elements could be part of such a method?*[2]

In Table 5.1 an overview of existing possible method fragments from literature on the topic of creating software from DEMO models is provided. In the exploratory case studies, model transformations or mappings from DEMO models to several technologies have been explored. While it might not be useful to combine all these transformations, each transformation individually can be considered a method fragment. The updated overview, including the newly created method fragments, is shown in Table 10.1. Additional research should provide insights in how these fragments can be combined to create a (situational) method for enterprise ontology-driven software development.

10.1.3 Reflection on Main Research Challenge

Main Research Challenge: *How to create a method for the development of enterprise software that answers the identified needs?*

With the answers to the theoretical and practical questions, a reflection on a reflection on the main research challenge is now possible. Having established what a method is (see Chapter 1), and that it is typically composed of fragments (see Chapter 5), an initial answer to the first part of the research challenge is provided. In order to make the method specifically address the identified needs, it has been shown that MDSD is an approach towards software development, aligned with the GSDP and the VSM (see Chapter 3), that provides the desired speed (need 3) and traceability (need 4). In order to make MDSD work, and to address the desired completeness of user requirements (need 1), a solution was found in using DEMO models as input models (see Chapter 4). In order to address the different layers of adaptability (need 2) and fill the gap towards enterprise implementation that DEMO does not address, the Enterprise Implementation Framework was created as additional input to the MDSD approach, that also enables explicit traceability from design decision to software implementation, further addressing need 4 (see Section 4.3). In order to further address the needs and possible shortcomings in

[2]In the original research question the more general term 'element' was used. As explained in Chapter 5, in the answer the term 'fragment' will be used.

Abstraction	Granularity	Perspective	
		Product	Process
Conceptual / abstract	Method Stage		GSDP MDSD, DEMO2services, DEMO2components, DEMO2NS, realization
	Model	DEMO aspect models, implementation model	OER
	Diagram	DEMO diagrams and tables	
	Concept	see DEMO metamodel (Figure 4.7), **OIVs** (Section 4.3)	
Conceptual / implementable	Stage		DEMO2WSBPEL, **DEMO2mockups** (Chapter 7), **DEMO2APIs** (Chapter 6), **DEMO2NS** (Chapter 8), **DEMO2lowcode** (Chapter 9)
	Model	DEMO exchange model (XML and JSON)	
Technical	Stage		CTP engine, **DEMO2OAS** (Chapter 6), **DEMO2Mendix** (Chapter 9)
	Model	OpenModeling, Simplified Modeling Platform, Sparx MDG	

Table 10.1: Method fragments from literature and from exploratory case studies (newly created in **bold**). Empty rows are left out.

the several choices made, target technologies for the approach were selected, viz., mockups, microservices, Normalized Systems, and low code (see Section 3.4).

The exploratory case studies in Part II have shown that such an approach is technically feasible and indeed addresses the needs. Moreover, these exploratory case studies have provided insights in the detailing of the needs as requirements for a complete method. Additionally, method fragments have been created that later can be combined into a complete method to generate software from the chosen enterprise models.

In terms of aspects of a method (see Figure 1.3), below the choices are summarized:

- the Way of Thinking is defined in Chapter 4, in which the WoT of DEMO is combined with a framework to describe enterprise implementation;
- the Way of Modeling is defined as the sum of DEMO ontological models and Organization Implementation Variables;
- the Way of Working is defined to be EMDSD that is a further refinement of MDSD, MBE in general and the GSDP for system development, along with the designed algorithms for conversion from these input models to some technology;
- the Way of Controlling is not specifically defined and it is suggested to use some existing WoC; and
- the Way of Supporting is defined as the set of modeling tools for DEMO and the created reference implementations for automated conversion.

This research has laid the foundation to start composing a 'complete' method for enterprise ontology-driven software development, including tools to support both researchers and practitioners. However, it can never be said that the results of this research are enough to indeed create such a method. Only by attempting to create a complete method, it will become clear whether this research has provided enough fragments or whether more research is still needed. With reference to the quote by Gandhi (see beginning of this chapter), the main result of this research is in the attempt to lay a foundation for a method, as it is impossible to (ever) claim that the creation of such a method has been finished.

10.2 Research Contributions

In this section the scientific and practical contributions of this research are outlined.

10.2.1 Scientific Contributions

This section details the scientific contributions.

Overview of MBE-related Notions

In Chapter 3 it has been shown that the development of enterprise software relies on the use of models. This is also known as MBE. In literature, several related approaches that use models to some extent to develop, engineer or architect systems in general or software specifically can be found. Sometimes, these notions are used interchangeably, while at other times these notions are used to denote something else. An overview of the different notions is provided and used to position this research. This overview could be used by other researchers in an attempt to find agreement on the terminology around MBE.

Comparison of Enterprise Modeling Techniques

In Chapter 4 an overview of available enterprise modeling techniques is provided, criteria are formulated to choose one that fits the chosen approach, i.e., EMDSD. While other overviews can be found in literature, this overview is considered a first attempt that uses multiple formalized criteria in order to choose a modeling technique for a specific purpose. This overview as well as the way of using it can be used by other researchers as well as practitioners.

Moreover, by tightly linking enterprise models to software creation, the possibility arises to use the software generated from these models to get feedback on the models. This may open up a new way to look at Participative Enterprise Modeling.

Enterprise Implementation Framework

As DEMO ontological models are abstracted from implementation, there is in fact a gap from these models to implementation. That gap is bridged in this research by creating the Enterprise Implementation Framework to capture enterprise implementation design decisions in terms of Organization Implementation Variables. This framework could be used by researchers to further define the notion of implementation model. This framework could be used by practitioners to actually design an enterprise implementation, linked to the ontological model of the enterprise. This framework can therefore also improve the adoption of DEMO as enterprise modeling technique, now that a way is provided to link DEMO to implementation.

Method Requirements

This research has resulted in a (revised) set of requirements for a method for the generation of enterprise software from enterprise ontological models. The requirements have laid the foundation to actually start composing the 'complete' method, including tools to support both the researcher and practitioner.

Method Fragments

The literature study has provided an overview of existing method fragments in order to compose a method for enterprise ontology-driven software development.

Moreover, the practical research has delivered some additional fragments, that are either technical or very close to being a technical fragment. Having fragments available was the last precondition for composing a 'complete' method, including supporting tools.

10.2.2 Practical Contributions

This section details the practical contributions.

Overview of Enterprise Modeling Techniques

In Chapter 4 an overview of available enterprise modeling techniques is provided, criteria are formulated to choose one that fits the chosen approach, i.e., EMDSD. While other overviews can be found in literature, this overview is considered a first attempt that uses multiple formalized criteria in order to choose a modeling technique for a specific purpose. This overview as well as the way of using it can be used by practitioners to choose a modeling technique for their own purpose, project or enterprise. The framework can easily be extended with additional modeling techniques and criteria.

Enterprise Implementation Framework

As DEMO ontological models are abstracted from implementation, there is in fact a gap from these models to implementation. That gap is bridged in this research by creating the Enterprise Implementation Framework to capture enterprise implementation design decisions in terms of Organization Implementation Variables. This framework could be used by (enterprise or business) architects to actually design an enterprise implementation, linked to the ontological model of the enterprise. They can also use it to consciously decide on and explicitly specify the required flexibility of the supporting software. This framework could be used by software developer to either challenge the aforementioned architects to further detail design decisions, or to capture the design decisions they used to implicitly hard code into software. In that way, at least the decisions are known and captured in a structured way. This framework could also improve the adoption of DEMO as enterprise modeling technique, now that is shown how it can be used in actual implementation and software development.

Method Fragments

The literature review showed that only a few technical fragments exist to support practitioners in creating software from DEMO models. The practical research has delivered some additional fragments, that are either technical or very close to being a technical fragment. These fragments have the possibility to support practitioners, and therefore could even improve the adoption of DEMO as it now has become easier to create working software from these models.

Completeness of the fragments is impossible to prove, and may in future be disproved. By using these fragments in practice, it can be validated whether

these fragments are enough, or whether additional ones are required. It is mostly likely, as shown in Section 1.2, that the set of method fragments will evolve by composing them into (situational) methods.

Addressing the Identified Needs

The exploratory case studies have shown that indeed the chosen approach helps in quickly creating enterprise software that supports the end users and is adaptable, in a traceable way. More specifically, it is adaptable to respond to expected changes on the enterprise level, i.e., adding or deleting a product or service, changing a business rule, or changing the implementation of a product or service. For changing the implementation, several possible expected types of change have been identified that can be used to consciously decide which ones to (easily) support with software, and which not – possibly resulting in a large redesign or rebuilt project if such a change needs to happen. It cannot be claimed that the quality of the resulting software is good, or better than with other approaches, but it has been shown that it *supports* in answering the identified needs, and therefore in achieving enterprise agility.

Moreover, these fragments, preferably supported by a tool that integrates them, could be an answer to legacy software, the tech talent shortage, and IT project failure. Currently, legacy applications need to be reverse engineering in order to understand their purpose and requirements, after which a new solution can be built. By starting from stable enterprise models, the processes and requirements that the software supports are known. Once a technology becomes outdated, a new mapping from the DEMO models to the new technology simply needs to be created so that all software can be regenerated towards the new technology. In that way, legacy migration is not an issue anymore.

By generating software from enterprise models, the enterprise model becomes the software and therefore can be considered a first step towards further integration of enterprise modeling and software development [165]. A direct consequence of this integration is that (more) software can be created with less highly skilled IT people as the models can be created by end users, architects or business analysts. This is in line with the trend of citizen development, that enables non-IT people to create software without (deep) knowledge of software development. This research has shown that the DEMO transaction kind could play a central role in creating software solution building blocks, that simply could be combined to support business process chains. This is currently known as *composable* architecture or composable software applications, a trend that also tries to provide an answer to the growing tech talent gap.

By applying an approach where software is generated from models with known quality attributes, and by reducing the amount of manual intervention in the process from models to working software to zero, the quality of the software is known as well. Project success or failure has many factors, of which expectations, in terms of requirements, budget and timelines, is a very important one. While it is hard to adjust or improve expectations and planning, this research has shown that it is possible to improve the actual implementation. Assuming that expectations

and planning do not change, the possibility arises to reduce project failure.

The designed approach can be considered a next level in 4GL (see Section 3.3.2). As the DEMO Action Model can be considered a constraint or logic model, the approach can be considered a step towards 5GL. More research is needed to see to what extent DEMO can be used for logic-based programming.

10.3 Impact

In this section the broader impact of this research will be outlined.

10.3.1 Research Goal and Results

The aim of this research is to work towards a method to generate enterprise software from enterprise models. The idea is that enterprises need to continuously adapt to stay ahead of competition, to keep their customers happy, and in order to comply with changing laws and regulations. Moreover, in order to support 'enterprise agility', it was identified that enterprise software should support their end users, should be adaptable, should be created quickly, and that it should be traceable how the end users' requirements are supported by the software. The only way to answer these needs, is by creating a structured method that guides its users on the path of software development for enterprises. Methods are not new, but due to advancements in technologies and enterprise modeling, this is the moment to create a new method.

In order to guarantee speed and traceability in the software development process, this research adopts an approach where software is fully generated from enterprise models. In order to guarantee that the right – and possibly all – user requirements are supported, DEMO models are used that only describe *what* a user needs and not *how* the user should be supported. As these models tend to leave out specific details, it is much easier to communicate and validate them. These models could even be used as reference models for a particular type of enterprise, e.g., a pizzeria or a bank. In order to prevent hard coding important details in the software, a framework was developed to capture additional design decisions. This framework also enables to explicitly design the required adaptability for enterprise software.

By means of exploratory case studies it has been shown that this approach is indeed feasible and addresses the identified needs. Moreover, four method elements were created that later can be combined into a complete method to generate software from enterprise models. This research thus has laid the foundations to create such a method.

10.3.2 Contributions

The results of this research show that the development of software can (and needs to) be drastically changed in order to support the so much desired enterprise agility. For scientific research it is required that the fields of enterprise modeling and software development are further integrated. A framework was developed

that can be used to bridge the gap from enterprise models to software, and it is suggested that it will be adopted in both scientific research and practical use cases. The practical research has shown that it is possible to use practical cases in a scientific context. Moreover, only by applying scientific research to practical cases, the real value can be validated. Scientific research in this area is encouraged to apply a practice-driven approach.

For enterprises, being public or commercial or some other, this research contributes to the improvement of the software development process, and to achieve enterprise agility in general. More specifically, with the designed approach, more software can be produced in less time, thus providing a way to lower the ever-growing IT budgets. As software can be changed more easily using the designed approach, it also offers new opportunities with a shorter time-to-market for new business innovations. As the approach uses high-level models that can be considered reference models, it becomes fairly easy to create specifically tailored software to similar organizations. Especially for municipalities and for financial, healthcare, and educational organizations, there is a big need to use reference models and standards while accommodating local differences in implementation.

Combining higher-level enterprise models and code generation has one more advantage: When technology advances, the only thing that is needed is to create a new transformation from the enterprise (meta)model to the new technology (metamodel). As soon as that mapping has been devised, all software can easily be regenerated from the enterprise models, without a detailed analysis of what the old legacy systems exactly do or should do.

Research shows that a lot of IT projects fail.[3] There are two dominant variables that define project failure or success: expectation (in terms of requirements, timelines and budget) and reality. If the gap is (too) big, a project is considered a failure. This research does not help to plan better for budgets or timelines. It does however help to improve and speed up implementation. This means that as long as plannings are being improved and expectations stay stable, this research can help in improving IT project success rate and save money by reducing project failure.

This research shows that it is possible to automate software development to a large extent, and use models that can typically be created in little time. As a result, less and less software developers are needed. The results of this research could therefore be part of a solution to deal with the tech shortage[4] as the designed approach enables persons without a background in IT to create software. This approach can therefore be considered an implementation of citizen development that enables and encourages non-IT people to create enterprise applications. The biggest difference between the current implementations of citizen development and this research is that current implementations rely on visualizing software components, whereas this research starts from the business level, i.e., processes, rules,

[3]See, e.g., a report from the Standish group: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf.

[4]See, e.g., https://www.gartner.com/en/newsroom/press-releases/2021-09-13-gartner-survey-reveals-talent-shortages-as-biggest-barrier-to-emerging-technologies-adoption.

and information. The designed approach can be considered an implementation of the paradigm of composable applications, where software solutions are being created by combining existing (functional) software components. If the DEMO transaction kind is considered a (functional) component, that is easily converted into working software, creating larger applications becomes nothing more than creating a chain of transaction kinds. This research could help in bringing citizen development and application composition to the next level, lowering the need for highly-skilled software developers.

Shortening the cycle from business idea to working software also has the potential to add value in the field of (Agile) Enterprise Design. By quickly implementing working software, the feedback cycles become shorter, and (potential) design flaws, in terms of business objectives, enterprise ontology, and enterprise implementation, can arise earlier in the process, reducing wasted investments.

10.3.3 Relevance

As the impact of this research is both scientific and practical, it is relevant for both researchers and practitioners. It is relevant for researchers in the area of software development as well as in the area of enterprise modeling and enterprise architecture. For practitioners, at first it may seem only relevant for 'IT people'. As software is an integral part of an enterprise and should be close to the end users, this research is also relevant for process owners, team leads, etc., that play a role in enterprise improvement and/or enterprise change (management) up to the highest management level. Any employee in an organization can use this research to show that software development can be improved and challenge the IT department to either use this research or come up with a better way. Moreover, tool builders could adopt this approach to further support the automatic creation of software.

The true value of this research can only be found by practical application. It is therefore necessary to apply it to more use cases, in small and big, and in public and commercial organizations. It is necessary to further validate and show the results of this research. Only then may such an approach be adopted by more and more enterprises, and may it become a new standard for software development. Adoption by one of the larger standardization entities and/or governmental agencies could speed up the adoption in practice.

"Es ist des Lernens kein Ende."
"There is no end to learning."
Robert Schumann, in Musikalische Haus- und Lebensregeln (1850)

11 Discussion

As learning and research is a never-ending process, in this chapter a critical reflection on this research is outlined, resulting in directions for further research. In Section 11.1 reflections on several choices made in this research are provided. In Section 11.2 the limitations of this research are discussed. Both sections are used as input to formulate future research directions in Section 11.3.

11.1 Reflections

In this section reflections on the choice MDSD as general approach, the chosen target technologies, the choice for DEMO as enterprise modeling technique, and to the research approach as a whole will be provided.

11.1.1 The Choice for MDSD

Model-driven Software Development was chosen as it seemed to address the identified needs 3 and 4. Applying MDSD also has downsides, including that it is considered complex, it is hard to include custom code, it moves complexity to the models, it needs input models with formal semantics (see Section 3.3.2). Regarding the need for formal semantics: it has been shown that by choosing a proper enterprise modeling technique, this concern can be dealt with.

Regarding the moving of complexity: it is argued that the complexity should be where it arises: Technical complexity should be in the software models, enterprise complexity should be in the enterprise models. As most complexity in software actually comes from complexity in the enterprise or its implementation, moving the complexity to the enterprise models is considered the only right approach. By doing so, the possibility opens up to make software development less

complex, and, more importantly, leave the design decisions where they belong, i.e., not deep in the software but on the enterprise level.

A related concern is that if the model is not good, the generated software also won't be good. This is in fact a concern for all model-driven software development approaches. At the same time, developing enterprise software without using models is considered a bad practice (see Section 1.1.3). The advantage of a model-driven approach is that if the result is not as expected, both the model and the mapping towards technology can be adapted, while retaining transparency in the mapping. Thus, although models have their limitations, they are still considered a must-have in software development. By setting models at the core of software development through model-driven code generation, the models even become a primary deliverable that should have the main focus before even thinking of software development.

Regarding the inclusion of custom code: by generating towards a low-code platform, it has been shown that it is possible to include custom code as these platforms are built to allow for integration of custom code. The NS expanders (code generators) have also shown that it is possible to combine generated and custom code in a controlled way.

Regarding its complexity: having experienced the exploratory case studies, it is hard to argue that it is not complex; creating a mapping from metamodel to metamodel is indeed complex, or at least hard. The creator of such a mapping needs to thoroughly know both the input and output (meta)model. On the other hand, creating such mappings is not new, and its complexity might just be in the eye of the beholder [502, 239].

Another concern in applying the MDSD approach might be that it is not desirable to automate everything. The working principle as applied in this research, based on the adopted WoT, is that every transaction kind or C-act should be supported by software, but does not need to be fully automated.[1] The created EIF does contain possibilities to distinguish between tasks that should be supported by IT and those that don't need support from IT. Another approach could be to simply leave out the transactions kinds and related entities and action rules in the models that drive the software generation. Still, scenarios can be thought of in which it is desired to manually transform an enterprise model into working software. This may mean that the designed EMDSD approach needs to be extended to an approach where it is possible to combine both manual and automated transformations.

While MDSD promises to close the gap between requirements and (software) implementation, there are also reasons to believe that this might not be possible at all. As shown in the GSDP (see Section 3.2), the step from functional requirements to implementation is the job of an engineer, that takes into account all requirements and tries to design a solution that best fits all or most requirements; such a mapping from functional requirements to solution is typically not a one-to-one mapping [98]. In this research, the functional requirements are merely

[1] By support is (only) meant that the end user can see the relevant details to make a decision and is guided along the process. By support is explicitly not meant that the software takes a decision fully automated.

ignored, and instead the gap from ontology or high-level conceptual model to implementation is bridged.[2] The high-level construction model clearly is not the same as user requirements, although it might be possible to create a, possibly not too complex, mapping between the two.

A final question to ask is 'is MDSD a *necessary* means to answer the needs'? And, to be honest, possibly it can be done without. Especially the rise of Large Language Models (LLMs) and generative AI [387][3] provides new possibilities to increase efficiency and accuracy in software development. At the same time, it is unlikely that these advancements will fully replace software developers, while it now enables both humans and computers to introduce implicit implementation design decisions into software. Only by using an algorithm to do a model transformation, traceability can be ensured. The exploratory case studies have shown that manually executing such an algorithm is possible, but also time-consuming and possibly error-prone. And if such an algorithm is on the level that it can be automated, computer execution will always beat a human in both speed and correctness of applying the algorithm. It seems that applying the MDSD approach was indeed a valid choice.

11.1.2 The Selection of Technologies

As shown in Section 3.4, four technologies were selected, that support addressing the needs and/or overcoming shortages of the MDSD approach. Mockups were chosen as a means to validate requirements (need 1), speed up development (need 3), and improve UI aspects. At the same time, they have been used to detect similarities and differences between similar enterprises (see Chapter 7), addressing need 2. Both microservices, Normalized Systems and low code were chosen as they offer a flexible landscape or solution, addressing need 2, while speeding up development (need 3).

The exploratory case studies were centered around the mapping from DEMO models to a specific technology (see Table 2.3). In each case, the target technology has delivered to meet or exceed the expectations. The selected technologies were also within the expertise area of the researcher, who played an active role in all exploratory case studies, as was needed to create the algorithms in the first place. Given that this research is in the exploratory phase (see Section 1.2), the selection of technologies was 'good enough' to take away (most of) the uncertainties as posed. At the same time, it is expected that other technologies could deliver similar results; it is thus not being said that the overall method should (only) incorporate the evaluated target technologies.

11.1.3 The Choice for DEMO

In order to choose an enterprise modeling technique that supports the main research challenge and chosen EMDSD approach, multiple criteria have been defined

[2] In the GSDP, this is known as technical design, see Section 3.2.2.
[3] Examples include ChatGPT.

(see Section 4.1.2). These criteria are derived from the general needs (see Section 1.1), specific needs from MDSD (see Section 3.3.2), requirements for a proper method fragment (see Figure 1.3), the perspectives that are considered important in enterprise modeling (see Section 4.1.1), and some usage and support factors. From the scoring of several enterprise modeling techniques it was concluded that DEMO has the best fit. If less or other perspectives were chosen, or if the presence of an underlying WoT was considered less important, or if the availability of tooling and examples were found more important, the chosen enterprise modeling technique could have been a different one. It could be beneficial to further verify the criteria and scoring, e.g., by means of a survey and/or an expert group decision session. Still, the advantages of DEMO have been proven by earlier research and these advantages are strengthened by this research.

One of the concerns in using DEMO is the lack of proper tooling. In the exploratory case studies the input file for the automated convertors were created manually. Although not ideal, this turned out to be workable.

Another concern in using DEMO is that is abstracted so far from implementation that additional techniques are required. In order to bridge the gap from enterprise ontology to implementation, the Enterprise Implementation Framework was developed (see Section 4.3). An alternative approach is to use the DEMO WoT and combine it with a modeling technique that includes more implementation details. For example, despite their incomparability to a large extent, ArchiMate and DEMO can be combined [135]. Other research has shown that DEMO models could be converted into BPMN [469, 353, 461, 70, 321, 183] or UML [33] models that have proven paths towards (automated) software development. However, combining these modeling techniques raises two new concerns: *a)* can these modeling techniques be properly combined or are there conflicts in their, possibly implicit, underlying WoT, and *b)* do modeling techniques like ArchiMate, BPMN and UML support the explicit and conscious decision-making process, allowing for adaptability on the three levels as defined in need 2?

As mentioned in Section 4.1.2, a Way of Controlling is not part of the designed method. DEMO does not include a WoC, nor do (most of) the other modeling techniques. Instead, modeling techniques mostly rely on existing WoCs such as PRINCE2, TOGAF, and Agile methods. It is thus necessary to further investigate what WoC best aligns to the chosen modeling technique.

Another possible approach to answer the needs is to use the DEMO WoT to create UML diagrams and then apply an MDA approach to generate software. As shown in Section 1.1.3 and Section 4.1, both MDA and UML have downsides that would hamper the method in answering the needs. Although not explicitly evaluated, using a modeling technique that is close to the (preferred) WoT, i.e., DEMO, is considered the best way to approach this research.

11.1.4 Research Approach and Case Selection

In Chapter 2 a nested ADR approach was defined, using exploratory case studies focused on a method fragment that can later be combined to form the base for a complete method. By performing multiple exploratory case studies, it became

possible to learn from the creation of one fragment, before creating another one. The fragments that have been created subsequently contain more detail in terms of enterprise implementation flexibility, and are subsequently geared more and more towards working software. Moreover, the exploratory case studies provided the necessary feedback on initial answers from the theoretical studies. On the method or research level the ADR approach has shown valuable.

For the exploratory case studies, it was important to design a fragment not only for a single enterprise. Multiple iterations have been conducted, where possible, to make sure the designed fragment works for multiple enterprises, adding relevance to the created fragment. While this aim was achieved, it can be reasoned more iterations are needed to really confirm the broad applicability. By performing multiple iterations that were enhanced by further theoretical research, rigor was added to the creation of the fragment. Together, applying ADR at the ECS or fragment level ensured grounded design science research, while solving real-world problems according to action research. While improvements are possible in the execution of individual ECSs, the nested ADR approach worked to achieve the research goal and to answer the research questions.

The selected real-world cases confirmed the relevance on both the method and the fragment level. It could be argued that the real-world cases that disproved the relevance were ignored. That is an inherent aspect of practice-driven research that is hard to avoid. At the same time, it might not be needed to show irrelevance as there are enterprises that see the relevance. The selected real-world cases were the source for designing and/or helped in evaluating a method fragment. The Social Housing case was used in the evaluation of two fragments, potentially showing that combining multiple fragments can be worthwhile. Additional research will be needed to confirm the findings of these exploratory case studies, to add to both relevance and rigor of the created fragments.

11.2 Limitations

A first limitation in this research is that the exploratory case studies were limited to academic cases and public or semi-public organizations. And although ECS 1 (Chapter 7) was performed on an international level, the other use cases are from the Netherlands. Apparently these were the organizations the researcher had access to and that were willing to cooperate in or support such an exploration. It does not immediately mean the results are only valid for Dutch public organizations only. However, additional validation, preferably with international commercial organizations would be preferred. This would also enable to better understand the possible complexity of supporting more OIVs in software (see Chapter 9).

Another limitation is that, while the aim for this research was to create technical fragments, only two technical fragments have been created, i.e., an implementation for the transformation of DEMO models to Mendix and one for the transformation of DEMO models to API specifications in OAS. The other fragments are considered detailed enough to be easily converted into a technical implemen-

tation, but weren't actually implemented with IT. The impact on the results of this research seems limited, as it was possible, but perhaps more labor intensive, to manually execute the algorithms to perform the transformations. In order to further validate these mappings and to support practical use, it is however recommended creating technical fragments as counterpart for the implementable but non-implemented fragments.

Moreover, for the technical implementations OAS and Mendix were chosen as target technologies. It could be that the choices were influenced by the possibilities of these target technologies, and that the designed approach does not work with other target platforms. Only by creating more implemented technical fragments with different target technologies and platforms, it can be verified whether the approach really works in practice.

While two algorithms have been automated and the two others are considered automatable, the claimed properties cannot be (mathematically) proven, they are only shown to be plausible by means of evaluation and/or reasoning. Further formalization of the fragments requires clear objectives and could be achieved, e.g., by applying formalization principles [443].

Finally, the exploratory case studies were mainly exploratory in the sense that it was never a goal to create working software, but more a means to show that such an approach is feasible and perhaps beneficial. While input from end users was used to create and validate the DEMO models, the output of the transformation, i.e., the working software was barely used by end users. It would be beneficial if the output software would be used by the actual end users to get more feedback on both the input models and the transformation to software.

11.3 Future Research

Based on the reflection on and limitations of this research, the following directions and opportunities for future research are formulated.

11.3.1 DEMO Support

During this research, it was noticed that the DEMO metamodel was not fully specified. In parallel to this research, research has been done to improve DEMO-SL [325]. However, the adapted metamodel does not seem to be fully incorporated into DEMO-SL yet and the exchange model includes not only DEMO concepts but also visualization aspects that are necessary for modeling tools but not for model transformation. These were also the main reasons to use an adapted metamodel and a cleaner JSON-based exchange format for the automed convertors. It would be beneficial if these efforts are integrated into one DEMO metamodel with a JSON-based exchange format that separates the DEMO concepts from its visualization.

Moreover, in order to further improve the adoption of the designed approach and DEMO in general, proper tooling is needed. Tooling could combine existing and newly created fragments, or is aimed for modeling purposes to create the input

for convertor(s) in the right format. This is a prerequisite to further explore the usage of DEMO in participative (enterprise) modeling and software creation.

11.3.2 Embedding in Existing Approaches

While the goal of this research was to contribute towards a method, the dream is still to have such a 'complete' method for enterprise ontology-driven software development. This dream implies not only having the right concepts, but also having the right tools, and having them embedded in a complete method that supports the complete software life cycle.

Embedding also means it is being adopted by a critical mass. In order to reach a bigger audience, integrating the designed approach with ArchiMate, BPMN and/or UML can be beneficial. Further formalization of the designed fragments is a necessary step in order to ensure integration. Moreover, the fragments should be integrated or aligned with a Way of Controlling. As stated earlier, there are some concerns regarding such an integration. More research should be done to investigate those concerns and see whether there are ways to overcome them.

Integrating approaches also means combining the better parts of one approach to overcome the 'flaws' of another approach. That also means that approaches or parts thereof should be compared somehow, in order to decide which fragments to use and which not. Research is needed to create comparison framework over enterprise model-based software development approaches.

As referred to earlier, while a risk with Large Language Models and (generative) Artificial Intelligence is that only more implementation design decisions are hard coded into software, there are also opportunities in combining such technologies with Enterprise Model-driven Software Development:

- (enterprise) modeling tools can be enhanced with AI so that similar advantages as in software development can be achieved, but on the level of enterprise modeling, in order to speed up and improve the input models for EMDSD;
- LLMs could transform text prompts into an enterprise model to allow users to create an input model for the automated convertors without having to learn the specific modeling technique;
- AI could be of support in optimizing implementation decisions; and
- it might be possible to use an LLM to detect (implicit) design decisions that are hard coded in software.

More research is needed to explore the potential of these upcoming technologies on the level of enterprise modeling and EMDSD.

11.3.3 Enterprise Agility

Enterprise agility is currently an important topic in Enterprise Engineering and Software Development. Although a first proposal is done how to measure enterprise agility, in practice it turns hard to objectively measure enterprise agility,

let alone compare different enterprises on their agility. From a practical point of view, it is often not desirable to actually create different implementations in the real world and compare their agility. Instead, numerous existing implementations need to be examined in order to be able to detect crucial design choices that support or hamper an enterprise's agility.

Measuring enterprise agility is also a prerequisite to define the constructs that are needed to support enterprise agility. Applying the Normalized Systems principles (see Section 3.4.3) to business processes has already been done [467], and the feasibility of normalized enterprises has been shown [218]. But what happens if the transactor role is used as core enterprise construct, with cross-cutting concerns around it? An initial set of such cross-cutting concerns has been proposed but not validated yet [97]. When comparing this set to the defined OIVs (see Section 4.3), it does raise the question whether all OIVs should be considered cross-cutting concerns, and whether that set is complete. More research is needed to find whether such a normalized construct for enterprises, fully compliant with the Normalized Systems principles, is feasible and improves enterprise agility. As a first step towards such a construct, it might be useful to formulate architecture principles related to the required enterprise agility, as shown in ECS2 (Chapter 7).

The normalized construct for enterprises may be comparable to the biological system that evolves around a stable core. What other lessons can be learned from biology in creating adaptable software and adaptable enterprises? It is clear that this is just the beginning of exploring concepts for enterprise agility, and that there is possibly much to learn from other research areas.

Bibliography

[1] C. Adelman, *Kurt Lewin and the Origins of Action Research*, Educational Action Research, (1993), pp. 7–24. 10.1080/0965079930010102.

[2] P. J. Ågerfalk, S. Brinkkemper, C. Gonzalez-Perez, B. Henderson-Sellers, F. Karlsson, S. Kelly, and J. Ralyté, *Modularization Constructs in Method Engineering: Towards Common Ground?*, in Situational Method Engineering: Fundamentals and Experiences, J. Ralyté, S. Brinkkemper, and B. Henderson-Sellers, eds., Boston, MA, 2007, Springer US, pp. 359–368. 10.1007/978-0-387-73947-2_27.

[3] P. J. Ågerfalk and B. Fitzgerald, *Exploring the Concept of Method Rationale: A Conceptual Tool to Understand Method Tailoring*, in Advanced Topics in Database Research, K. Siau, ed., vol. 5, IGI Global, 2006, pp. 63–78. 10.4018/978-1-59140-935-9.ch004.

[4] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers : principles, techniques, and tools*, Addison-Wesley, Reading, Massachusetts, 1985.

[5] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, *Model traceability*, IBM Systems Journal, 45 (2006), pp. 515–526. 10.1147/sj.453.0515.

[6] R. Akif and H. Majeed, *Issues and Challenges in Scrum Implementation*, International Journal of Scientific & Engineering Research, 3 (2012).

[7] I. K. Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, *A Model-Driven Architecture for Automated Deployment of Microservices*, Applied Sciences, 11 (2021). 10.3390/app11209617.

[8] A. Albani and J. L. G. Dietz, *Enterprise ontology based development of information systems*, International Journal of Internet and Enterprise Management, 7 (2011). 10.1504/IJIEM.2011.038382.

[9] S. W. Ambler, *Agile Model Driven Development Is Good Enough*, IEEE Software, 20 (2003), pp. 71–73. 10.1109/MS.2003.1231156.

[10] D. Ameller, *Considering Non-Functional Requirements in Model-Driven Engineering*, mathesis, Universitat Politècnica de Catalunya, Jan. 2011.

[11] K. P. Arnett and M. C. Jones, *Programming Languages: Today and Tomorrow*, Journal of Computer Information Systems, 33 (1993), pp. 77–81. 10.1080/08874417.1993.11646550.

[12] L. Arts, M. K. Chmarra, and T. Tomiyama, *Modularization Method for Adaptable Products*, in roceedings of the ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 4, 2008, pp. 193–202. 10.1115/DETC2008-49338.

[13] U. Assmann, S. Zschaler, and G. Wagner, *Ontologies, Meta-models, and the Model-Driven Paradigm*, in Ontologies for Software Engineering and Software Technology, C. Calero, F. Ruiz, and M. Piattini, eds., Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 249–273. 10.1007/3-540-34518-3_9.

[14] J. L. Austin, *How to do things with words*, William James Lectures, Oxford University Press, 1962.

[15] D. Aveiro and V. Freitas, *A new Action Meta-Model and Grammar for a DEMO based low-code platform rules processing engine*, in Advances in Enterprise Engineering XIV, 2023.

[16] D. Aveiro and D. Pinto, *A Case Study Based New DEMO Way of Working and Collaborative Tooling*, in 2013 IEEE 15th Conference on Business Informatics, 2013, pp. 21–26. 10.1109/CBI.2013.12.

[17] D. E. Avison and G. Fitzgerald, *Information Systems Development: Methodologies, Techniques and Tools*, McGraw-Hill, second ed., 1995.

[18] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen, *Action research*, Communications of the ACM, 42 (1999), pp. 94–97.

[19] M. Azarm-Daigle, C. Kuziemsky, and L. Peyton, *A Review of Cross Organizational Healthcare Data Sharing*, Procedia Computer Science, 63 (2015), pp. 425–432. 10.1016/j.procs.2015.08.363.

[20] C. L. B. Azevedo, M.-E. Iacob, J. P. A. Almeida, M. van Sinderen, L. Ferreira Pires, and G. Guizzardi, *Modeling resources and capabilities in enterprise architecture: A well-founded ontology-based proposal for ArchiMate*, Information Systems, 54 (2015), pp. 235–262. 10.1016/j.is.2015.04.008.

[21] T. G. Badger, *Action research, change and methodological rigour*, Nursing Management, 8 (2000), pp. 201–207. 10.1046/j.1365-2834.2000.00174.x.

[22] K. Bakshi, *Microservices-based software architecture and approaches*, in 2017 IEEE Aerospace Conference, 2017, pp. 1–8. 10.1109/AERO.2017.7943959.

[23] L. Balmelli, D. Brown, M. Cantor, and M. Mott, *Model-driven systems development*, IBM Systems Journal, 45 (2006), pp. 569–585. 10.1147/sj.453.0569.

[24] J. P. Bansler and K. Bødker, *A Reappraisal of Structured Analysis: Design in an Organizational Context*, ACM Transactions on Information Systems, 11 (1993), pp. 165–193. 10.1145/130226.148055.

[25] I. Baroi and S. De, *A Novel Application of Neuromarketing for Designing User Interface Mockups to Enhance User Experience in Software Development*, in 2021 IEEE 2nd International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET), 2021. 10.1109/TEMSMET53515.2021.9768683.

[26] N. M. J. Basha, S. A. Moiz, and M. Rizwanullah, *Model Based Software Development: Issues & Challenges*, International Journal of Computer Science and Informatics, 3 (2013). 10.47893/IJCSI.2013.1123.

[27] R. L. Baskerville, B. Ramesh, L. Levine, and J. Pries-Heje, *High-Speed Software Development Practices: What Works, What Doesn't*, IT Professional, 8 (2006), pp. 29–36. 10.1109/MITP.2006.86.

[28] R. L. Baskerville and A. T. Wood-Harper, *A critical perspective on action research as a method for information systems research*, Journal of Information Technology, 11 (1996), pp. 253–246. 10.1080/026839696345289.

[29] ——, *Diversity in information systems action research methods*, European Journal of Information Systems, 7 (1998), pp. 90–107. 10.1057/palgrave.ejis.3000298.

[30] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, *Using structural and semantic measures to improve software modularization*, Empirical Software Engineering, 18 (2013), pp. 901–932. 10.1007/s10664-012-9226-8.

[31] P. Baxter and S. Jack, *Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers*, The Qualitative Report, 13 (2008), pp. 544–559. 10.46743/2160-3715/2008.1573.

[32] S. Beer, *The Viable System Model: Its Provenance, Development, Methodology and Pathology*, Journal of the Operational Research Society, 35 (1984), pp. 7–25. 10.1057/jors.1984.2.

[33] I. Begetis, *Combining Design and Engineering Methodology for Organizations with the Rational Unified Process*, Master's thesis, Delf University of Technology, 2010.

[34] A. E. Bell, *Death by UML Fever*, Queue, 2 (2004).

[35] P. Bertolazzi, C. Krusich, and M. Missikoff, *An Approach to the Definition of a Core Enterprise Ontology: CEO*, Computer Science, (2001).

[36] N. Bevan, *Quality in use: Meeting user needs for quality*, Journal of Systems and Software, 49 (1999), pp. 89–96. 10.1016/S0164-1212(99)00070-9.

[37] M. Bexiga, S. Garbatov, and J. C. Seco, *Closing the Gap between Designers and Developers in a Low Code Ecosystem*, in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, New York, NY, USA, Oct. 2020, pp. 1–10. 10.1145/3417990.3420195.

[38] S. Beydeda, M. Book, and V. Gruhn, eds., *Model-Driven Software Development*, Springer Berlin, Heidelberg, 2005. 10.1007/3-540-28554-7.

[39] J. Bézivin, *On the unification power of models*, Software & Systems Modeling, 4 (2005), pp. 171–188. 10.1007/s10270-005-0079-0.

[40] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow, *Model Transformations? Transformation Models!*, in Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds., vol. 4199 of Lecture Notes in Computer Science, Springer, Oct. 2006, pp. 440–453. 10.1007/11880240_31.

[41] J. Bézivin and O. Gerbé, *Towards a precise definition of the OMG/MDA framework*, in Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), 2001, pp. 273–280. 10.1109/ASE.2001.989813.

[42] N. Bieberstein, S. Bose, L. Walker, and A. Lynch, *Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals*, IBM Systems Journal, 44 (2005), pp. 691–708. 10.1147/sj.444.0691.

[43] M. Biehl, *Literature Study on Model Transformations*, tech. rep., Royal Institute of Technology Stockholm, Sweden, July 2010.

[44] E. Bigdeli, M. Motadel, A. Toloie Eshlaghy, and R. Radfar, *A dynamic model of effective factors on Agile business-IT alignment*, Kybernetes, 49 (2020), pp. 2521–2546. 10.1108/K-05-2019-0358.

[45] A. Bischof and L. Blessing, *Guidelines for the Development of Flexible Products*, in DS 48: Proceedings DESIGN 2008, the 10th International Design Conference, D. Marjanovic, M. Storga, N. Pavkovic, and N. Bojcetic, eds., 2008, pp. 289–300.

[46] J. D. Blackburn, G. D. Scudder, and L. N. Van Wassenhove, *Improving speed and productivity of software development: a global survey of software developers*, IEEE Transactions on Software Engineering, 22 (1996), pp. 875–885. 10.1109/32.553636.

[47] J. K. Blundell, M. L. Hines, and J. Stach, *The measurement of software design quality*, Annals of Software Engineering, 4 (1997), pp. 235–255. 10.1023/A:1018914711050.

[48] A. C. Bock and U. Frank, *MEMO GoalML: A Context-Enriched Modeling Language to Support Reflective Organizational Goal Planning and Decision Processes*, in Conceptual Modeling, 35th International Conference, ER 2016, vol. 9974 of Lecture Notes in Computer Science, 2016, pp. 515–529.

[49] ——, *Multi-Perspective Enterprise Modelling - Conceptual Foundation and Implementation with ADOxx*, in Domain-Specific Conceptual Modeling, D. Karagiannis, H. Mayr, and J. Mylopoulos, eds., Springer, Cham, 2016, pp. 241–267. 10.1007/978-3-319-39417-6_11.

[50] ——, *In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms*, in 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 57–66. 10.1109/MODELS-C53483.2021.00016.

[51] A. D. Booth and K. H. V. Britten, *Coding for A.R.C.*, tech. rep., Institute for Advanced Study, Princeton, 1947.

[52] D. Bork and H.-G. Fill, *Formal Aspects of Enterprise Modeling Methods: A Comparison Framework*, in 47th Hawaii International Conference on System Sciences, IEEE, Jan. 2014, pp. 3400–3409. 10.1109/HICSS.2014.422.

[53] W. Borst, *Construction of Engineering Ontologies*, PhD thesis, Institute forTelematica and Information Technology, University of Twente, 1997.

[54] J. Bosch, *On the Development of Software Product-Family Components*, in Proceedings of the 3rd International Conference on Software Product Lines (SPLC), Springer LNCS, 2004. 10.1007/b100081.

[55] A. Bragança and R. J. Machado, *Model Driven Development of Software Product Lines*, in Proceedings of the 6th International Conference on Quality of Information and Communications Technology (QUATIC), IEEE, 2007, pp. 199–203. 10.1109/QUATIC.2007.25.

[56] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, second ed., Sept. 2017. 10.2200/S00441ED1V01Y201208SWE001.

[57] S. Brinkkemper, *Formalisation of Information Systems Modelling*, PhD thesis, Radboud University Nijmegen, June 1990.

[58] ——, *Method engineering: Engineering of information systems development methods and tools*, Information & Software Technology, 38 (1996), pp. 275–280. 10.1016/0950-5849(95)01059-9.

[59] S. Brinkkemper, M. Saeki, and F. Harmsen, *Assembly techniques for method engineering*, in Advanced Information Systems Engineering, B. Pernici and C. Thanos, eds., Berlin, Heidelberg, 1998, Springer Berlin Heidelberg, pp. 381–400.

[60] A. W. Brown, *Model driven architecture: Principles and practice*, Software and Systems Modeling, 3 (2004), pp. 314–324. 10.1007/s10270-004-0061-2.

[61] T. Brown, *Design Thinking*, Harvard business review, 86 (2008), pp. 84–92,141.

[62] M. Broy and M. V. Cengarle, *UML formal semantics: lessons learned*, Software and Systems Modeling, 10 (2011), pp. 441–116. 10.1007/s10270-011-0207-y.

[63] A. Bryman and E. Bell, *Business Research Methods*, Oxford University Press, 2011.

[64] I. Bucena and M. Kirikova, *Simplifying the DevOps Adoption Process*, in Joint Proceedings of the BIR 2017 pre-BIR Forum, Workshops and Doctoral Consortium co-located with 16th International Conference on Perspectives in Business Informatics Research (BIR 2017), Copenhagen, Denmark, August 28 - 30, 2017, B. Johansson, ed., vol. 1898 of CEUR Workshop Proceedings, CEUR-WS.org, 2017. http://ceur-ws.org/Vol-1898/paper14.pdf.

[65] C. Bürger, S. Karol, and C. Wende, *Applying Attribute Grammars for Metamodel Semantics*, in Proceedings of the International Workshop on Formalization of Modeling Languages, FML '10, New York, NY, USA, 2010, Association for Computing Machinery. 10.1145/1943397.1943398.

[66] T. Bush, *What Is The Difference Between APIs and Microservices?* Online, Jan. 2019. Accessed 2021-Nov-18. https://nordicapis.com/what-is-the-difference-between-apis-and-microservices/.

[67] Business Agility Institute, *Domains of Business Agility*. Online, 2020. Accessed 2022-Nov-08. https://api.businessagility.institute/storage/files/download-publications/bai-domains-of-business-agility-digital.pdf.

[68] J. Cabot, *Executable models vs code-generation vs model interpretation*. Online, Aug. 2010. Accessed 2022-May-17. https://modeling-languages.com/executable-models-vs-code-generation-vs-model-interpretation-2/.

[69] ——, *Positioning of the Low-Code Movement within the Field of Model-Driven Engineering*, in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, New York, NY, USA, 2020, Association for Computing Machinery. https://doi.org/10.1145/3417990.3420210.

[70] A. Caetano, A. Assis, and J. Tribolet, *Using Business Transactions to Analyse the Consistency of Business Process Models*, in Proceedings of CONFENIS 2011, C. Møller, ed., CIP Working Papers, Center for Industrial Production, Aalborg University, 2011.

[71] ——, *Using DEMO to analyse the consistency of business process models*, in Advances in Enterprise Information Systems II, CRC Press, June 2012, pp. 133–146. 10.1201/b12295-17.

[72] D. Card, G. T. Page, and F. E. McGarry, *Criteria for Software Modularization*, in Proceedings of the 8th international conference on Software engineering, M. M. Lehman, H. Hünke, and B. W. Boehm, eds., IEEE Computer Society, 1985, pp. 372–377.

[73] P. P. Cardoso Castro, *The viable system model as a framework to guide organisational adaptive response in times of instability and change*, International Journal of Organizational Analysis, 27 (2019), pp. 289–307. 10.1108/IJOA-01-2018-1334.

[74] M. Carr and J. Verner, *Prototyping and Software Development Approaches*, tech. rep., Department of Information Systems, City University of Hong Kong, 1997.

[75] J. P. Cavano and J. McCall, *A framework for the measurement of software quality*, SIGMETRICS Perform. Evaluation Rev., 7 (1978), pp. 133–139.

[76] M. R. V. Chaudron, A. Fernandes-Saez, R. Hebig, T. Ho-Quang, and R. Jolak, *Diversity in UML Modeling Explained: Observations, Classications and Theorizations*, in 44th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2018, vol. 10706 of Lecture Notes in Computer Science, 2018, pp. 47–66. 10.1007/978-3-319-73117-9_4.

[77] F. Chauvel and J.-M. Jézéquel, *Code Generation from UML Models with Semantic Variation Points*, in Model Driven Engineering Languages and Systems (MODELS 2005), L. Briand and C. Williams, eds., vol. 3713 of Lecture Notes in Computer Science, Berlin, Heidelberg, 2005, Springer Berlin Heidelberg, pp. 54–68. 10.1007/11557432_5.

[78] L. Chen, *Continuous Delivery: Huge Benefits, but Challenges Too*, IEEE Software, 32 (2015), pp. 50–54. 10.1109/MS.2015.27.

[79] P. Chen, *The Entity-Relationship Model - Toward a Unified View of Data*, ACM Transactions on Database Systems, 1 (1976), pp. 9–36.

[80] S. Ciraci and P. van den Broek, *Evolvability as a quality attribute of software architectures*, Journal of Physics: Conference Series, (2006).

[81] R. Cole, S. Purao, M. Rossi, and M. K. Sein, *Being proactive: where action research meets design research*, in ICIS 2005 Proceedings, D. Avison, D. Galletta, and J. I. DeGross, eds., 2005, pp. 325–336.

[82] J. Collis and R. Hussey, *Business Research: A Practical guide for undergraduate and postgraduate students*, Palgrave Macmillan, second ed., 2003.

[83] Computer Economics, *IT Spending and Staffing Benchmarks 2022/2023*, tech. rep., Avasant Research, 2022.

[84] K. Conboy and B. Fitzgerald, *Toward a Conceptual Framework of Agile Methods: A Study of Agility in Different Disciplines*, in Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research, WISER '04, New York, NY, USA, 2004, ACM, pp. 37–44. 10.1145/1029997.1030005.

[85] S. Cook, *Domain-specific modeling and model driven architecture*, MDA Journal, 9 (2004), pp. 2–10.

[86] C. Corea, M. Fellmann, and P. Delfmann, *Ontology-Based Process Modelling - Will We Live to See It?*, in Conceptual Modeling, A. Ghose, J. Horkoff, V. E. Silva Souza, J. Parsons, and J. Evermann, eds., vol. 13011 of Lecture Notes in Computer Science, Cham, 2021, Springer International Publishing, pp. 36–46. 10.1007/978-3-030-89022-3_4.

[87] S. Cronholm and P. J. Ågerfalk, *On the Concept of Method in Information Systems Development*, in Proceedings of the 22nd Information Systems Research in Scandinavia (IRIS 22): Enterprise Architectures for Virtual Organisations, T. K. Käkölä, ed., vol. 1, 1999, pp. 229–236.

[88] P. B. Crosby, *Quality Is Free: The Art of Making Quality Certain*, Mentor book, McGraw-Hill, 1979.

[89] S. Crowe, K. Cresswell, A. Robertson, G. Huby, A. Avery, and A. Sheikh, *The case study approach*, BMC Med Res Methodol, 11 (2011). 10.1186/1471-2288-11-100.

[90] K. Czarnecki and S. Helsen, *Feature-Based Survey of Model Transformation Approaches*, IBM Systems Journal, 45 (2006), pp. 621–645. 10.1147/sj.453.0621.

[91] M. Daneva and R. Wieringa, *Cost estimation for cross-organizational ERP projects: research perspectives*, Software Quality Journal, 16 (2008), pp. 459–481. 10.1007/s11219-008-9045-8.

[92] F. Daoudi and S. Nurcan, *A framework to evaluate methods' capacity to design flexible business processes*, in 6th International Workshop on Business Process Modeling, vol. 12, Porto, Portugal, 2006, pp. 1–8. 10.1002/spip.308.

[93] R. A. D'aveni, *Hypercompetition*, Simon & Schuster, 2010.

[94] A. M. Davis, *Fifteen principles of software engineering*, IEEE Software, 11 (1994), pp. 94–96. 10.1109/52.329409.

[95] M. Dawson, D. Burrell, E. Rahim, and S. Brewster, *Integrating Software Assurance into the Software Development Life Cycle (SDLC)*, Journal of Information Systems Technology and Planning, 3 (2010), pp. 49–53.

[96] B. de Bruin, A. Verschut, and E. Wierstra, *Systematic Analysis of Business Processes*, Knowledge & Process Management, 7 (2000), pp. 87–96. 10.1002/1099-1441(200004/06)7:2<87::AID-KPM71>3.0.CO;2-4.

[97] P. De Bruyn and H. Mannaert, *Towards Applying Normalized Systems Concepts to Modularity and the Systems Engineering Process*, in ICONS 2012, The Seventh International Conference on Systems, 2012, pp. 59–66.

[98] P. De Bruyn, D. Van Nuffel, P. Huysmans, and H. Mannaert, *Towards Functional and Constructional Perspectives on Business Process Patterns*, in ICSEA 2011 : The Sixth International Conference on Software Engineering Advances, Oct. 2011, pp. 459–464.

[99] P. De Bryun, H. Mannaert, J. Verelst, and P. Huysmans, *Enabling Normalized Systems in Practice - Exploring a Modeling Approach*, Business & Information Systems Engineering, 60 (2018), pp. 55–67. 10.1007/s12599-017-0510-4.

[100] R. de Feijter, R. van Vliet, E. Jagroep, S. Overbeek, and S. Brinkkemper, *Towards the adoption of DevOps insoftware product organizations: A maturitymodel approach*, Tech. Rep. UU-CS-2017-009, Utrecht University, May 2017.

[101] J. de Jong, *A Method for Enterprise Ontology based Design of for Enterprise Information Systems*, PhD thesis, TU Delft, 2013.

[102] J. de Jong and J. L. G. Dietz, *Understanding the realization of organizations*, in Advances in Enterprise Engineering IV, W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, A. Albani, and J. L. G. Dietz, eds., vol. 49 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2010, pp. 31–49.

[103] S. de Kinderen, K. Gaaloul, and H. A. Proper, *On Transforming DEMO Models to ArchiMate*, in Enterprise, Business-Process and Information Systems Modeling, I. Bider, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, and S. Wrycza, eds., Springer Berlin Heidelberg, 2012, pp. 270–284. 10.1007/978-3-642-31072-0_19.

[104] L. de Laat, M. Op 't Land, and M. R. Krouwel, *Supporting Goal-oriented Organizational Implementation - Combining DEMO and Process Simulation in a Practice-tested Method*, in EEWC 2016: Advances in Enterprise Engineering X, D. Aveiro, R. Pergl, and D. Gouveia, eds., vol. 252 of Lecture Notes in Business Information Processing, Springer, 2016, pp. 19–33. 10.1007/978-3-319-39567-8_2.

[105] De Rijksoverheid, *Algemene wet bestuursrecht*, 2022. Accessed on 2022-Dec-13. https://wetten.overheid.nl/BWBR0005537/2022-11-05.

[106] M. De Vries, *DEMO and the Story-Card Method: Requirements Elicitation for Agile Software Development at Scale*, in PoEM 2018: The Practice

of Enterprise Modeling, R. Buchmann, D. Karagiannis, and M. Kirikova, eds., vol. 335 of Lecture Notes in Business Information Processing, Springer, 2018, pp. 138–153. 10.1007/978-3-030-02302-7_9.

[107] C. Décosse, W. A. Molnar, and H. A. Proper, *What Does DEMO Do? A Qualitative Analysis about DEMO in Practice: Founders, Modellers and Beneficiaries*, in Proceedings of the 4th Enterprise Engineering Working Conference (EEWC 2014), Funchal, Madeira, vol. 174 of Lecture Notes in Business Information Processing, 2014, pp. 16–30. 10.1007/978-3-319-06505-2_2.

[108] A. Delgado and D. Calegari, *Towards integrating BPMN 2.0 with CMMN and DMN standards for flexible business process modeling*, in 22nd Conferencia Iberoamericana en Software Engineering (CIbSE 2019), Curran Associates, Inc., Apr. 2019, pp. 697–704.

[109] T. DeMarco, *Structured Analysis and System Specification*, Prentice Hall, 1979.

[110] J. Den Haan, *An Enterprise Ontology based approach to Model-Driven Engineering*, Master's thesis, TU Delft, Oct. 2009. https://repository.tudelft.nl/islandora/object/uuid:e2093132-9db7-4cba-bc68-9355f93cb9e3.

[111] V. Desai, Y. Koladia, and S. Pansambal, *Microservices: Architecture and Technologies*, International Journal for Research in Applied Science & Engineering Technology (IJRASET), 8 (2020). 10.22214/ijraset.2020.31979.

[112] M. S. Deutsch and R. R. Willis, *Software Quality Engineering: A Total Technical and Management Approach*, Prentice Hall, 1988.

[113] G. Dévai, M. Karácsony, B. Németh, R. Kitlei, and T. Kozsik, *UML Model Execution via Code Generation*, in EXE@MoDELS, 2015.

[114] V. Devedzić, *Understanding Ontological Engineering*, Communications of the ACM, 45 (2002), pp. 136–144. 10.1145/505248.506002.

[115] P. Di Francesco, P. Lago, and I. Malavolta, *Architecting with Microservices: A Systematic Mapping Study*, Journal of Systems and Software, (2019).

[116] D. Di Ruscio, D. Kolovos, J. Lara, A. Pierantonio, M. Tisi, and M. Wimmer, *Low-code development and model-driven engineering: Two sides of the same coin?*, Software and Systems Modeling, 21 (2022), pp. 437–446. 10.1007/s10270-021-00970-2.

[117] J. Dietz, E. van Dipten, M. Krouwel, P. Kuipers, T. de Mik, and I. Theuwissen-Krol, *Begrijpen en Maken van Essentiële Modellen*, Werken met DEMO (WmD), Enterprise Engineering Institute, Nov. 2021. Dutch. https://ee-institute.org/demo/werken-met-demo/.

[118] J. L. G. Dietz, *Deriving Use Cases from Business Process Models*, in Conceptual Modeling - ER 2003, I.-Y. Song, S. Liddle, T.-W. Ling, and P. Scheuermann, eds., vol. 2813 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 131–143. 10.1007/978-3-540-39648-2_13.

[119] ——, *Enterprise Ontology — Theory and methodology.*, Springer, 2006.

[120] ——, *Architecture – Building strategy into design*, Netherlands Architecture Forum, Academic Service – SDU, The Hague, The Netherlands, 2008.

[121] ——, *The Essence of Organisation - an Introduction to Enterprise Engineering*, Sapio, 2013. http://www.sapio.nl.

[122] ——, *The DEMO Specification Language v4.7*, tech. rep., Enterprise Engineering Institute, Apr. 2022. https://ee-institute.org/download/demo-specification-language-4-7-1/.

[123] J. L. G. Dietz and J. A. P. Hoogervorst, *Enterprise Ontology and Enterprise Architecture – how to let them evolve into effective complementary notions.*, GEAO Journal of Enterprise Architecture, 2007, 1 (2007).

[124] J. L. G. Dietz and J. B. F. Mulder, *Enterprise Ontology — A Human-Centric Approach to Understanding the Essence of Organisation*, The Enterprise Engineering Series, Springer, Cham, 2020. 10.1007/978-3-030-38854-6.

[125] J. L. G. Dietz, M. Op 't Land, M. R. Krouwel, and J. B. F. Mulder, *Enterprise Design*, Springer, 2024. Forthcoming.

[126] R. M. Dijkman, M. Dumas, and C. Ouyang, *Semantics and analysis of business process models in BPMN*, Information and Software Technology, 50 (2008), pp. 1281–1294. 10.1016/j.infsof.2008.02.006.

[127] E. W. Dijkstra, *Notes on Structured Programming*, tech. rep., Technological University Eindhoven, Department of Mathematics, Apr. 1970. Vol. 70-WSK-03.

[128] R. Dove, *Agile Enterprise Cornerstones: Knowledge, Values, and Response Ability*, in Business Agility and Information Technology Diffusion, R. L. Baskerville, L. Mathiassen, J. Pries-Heje, and J. I. DeGross, eds., vol. 180 of IFIP International Federation for Information Processing, Boston, MA, 2005, Springer US, pp. 313–330. 10.1007/0-387-25590-7_20.

[129] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: yesterday, today, and tomorrow*, Present and Ulterior Software Engineering, (2017), pp. 195–216. 10.1007/978-3-319-67425-4_12.

[130] E. Eessaar, *On Applying Normalized Systems Theory to the Business Architectures of Information Systems*, Baltic Journal of Modern Computing, 2 (2014), pp. 132–149.

[131] K. M. Eisenhardt, *Building theories from case study research*, Academy of management review, 14 (1989), pp. 532–550.

[132] D. W. Embley, B. D. Kurtz, and S. N. Woodfield, *Object-Oriented Systems Analysis, A Model-Driven Approach*, Yourdon Press, 1992.

[133] R. Espejo and A. Reyes, *Organizational Systems: Managing Complexity with the Viable System Model*, Springer, 2011.

[134] R. W. Ettema, *Using triangulation in lean six sigma to explain quality problems – An enterprise engineering perspective*, PhD thesis, Radboud University, 2016.

[135] R. W. Ettema and J. L. G. Dietz, *ArchiMate and DEMO - Mates to Date?*, in Advances in Enterprise Engineering III. CIAO! EOMAS 2009, A. Albani, J. Barjis, and J. L. G. Dietz, eds., vol. 34 of Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg, 2009. 10.1007/978-3-642-01915-9_13.

[136] M. W. Evans and J. J. Marciniak, *Software Quality Assurance & Management*, John Wiley & Sons, 1987.

[137] J. Evermann and Y. Wand, *Towards Ontologically Based Semantics for UML Constructs*, in Conceptual Modeling – ER 2001, H. S.Kunii, S. Jajodia, and A. Sølvberg, eds., vol. 2224 of Lecture Notes in Computer Science, Berlin, Heidelberg, 2001, Springer Berlin Heidelberg, pp. 354–367. 10.1007/3-540-45581-7_27.

[138] R. A. Falbo, G. Guizzardi, K. Duarte, and A. Natali, *Developing software for and with reuse: an ontological approach*, in ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications, International Association for Computer and Information Science (ACIS), June 2002, pp. 311–316.

[139] D. Farinelli and D. McAllister, *API vs. Microservices: A Microservice Is More Than Just an API*. Online, Oct. 2019. Accessed 2021-Nov-11. https://dzone.com/articles/api-vs-microservices-a-microservice-is-more-than-j.

[140] D. Farley, *The Problem With Microservices*. Online, Oct. 2020. Accessed 2021-Nov-18. https://www.youtube.com/watch?v=zzMLg3Ys5vI.

[141] J.-M. Favre, *Foundations of Model (Driven) (Reverse) Engineering : Models - Episode I: Stories of The Fidus Papyrus and of The Solarus*, in Language Engineering for Model-Driven Software Development, J. Bezivin and R. Heckel, eds., vol. 4101 of Dagstuhl Seminar Proceedings (DagSemProc),

Dagstuhl, Germany, 2004, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 1–31. https://drops.dagstuhl.de/opus/volltexte/2005/13, 10.4230/DagSemProc.04101.8.

[142] ——, *Megamodelling and Etymology*, in Transformation Techniques in Software Engineering, J. R. Cordy, R. L"ammel, and A. Winter, eds., no. 05161 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006, Internationales Begegnungs- und Forschungszentrum f"ur Informatik (IBFI), Schloss Dagstuhl, Germany. http://drops.dagstuhl.de/opus/volltexte/2006/427.

[143] M. Fayad and M. P. Cline, *Aspects of Software Adaptability*, Communications of the ACM, 39 (1996), pp. 58–59.

[144] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, SEI Series in Software Engineering, Pearson Education, 2012. https://books.google.nl/books?id=T_nOltBoX64C.

[145] J. Ferreira, J. Noble, and R. Biddle, *Agile Development Iterations and UI Design*, in Agile 2007, 2007, pp. 50–58. 10.1109/AGILE.2007.8.

[146] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. Online, June 1999. Accessed 2022-Apr-01. http://www.rfc.net/rfc2616.html.

[147] R. R. Filho, *Emergent Software Systems*, PhD thesis, Lancaster University, UK, 2018.

[148] M. Filipović, Ž. Vuković, I. Dejanović, and G. Milosavljević, *Rapid Requirements Elicitation of Enterprise Applications Based on Executable Mockups*, Applied Sciences, 11 (2021). 10.3390/app11167684.

[149] H.-G. Fill and D. Karagiannis, *On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform*, Enterprise Modelling and Information Systems Architecture, 8 (2013), pp. 4–24.

[150] C. Fischer, R. Winter, and F. Wortmann, *Design Theory*, Business & Information Systems Engineering, 2 (2010), pp. 387–390. 10.1007/s12599-010-0128-2.

[151] B. Fitzgerald, N. L. Russo, and E. Stolterman, *Information Systems Development: Methods-in-Action*, McGraw-Hill, London, June 2002.

[152] B. Flyvbjerg, *Five Misunderstandings About Case-Study Research*, Qualitative Inquiry, 12 (2006), pp. 219–245. 10.1177/1077800405284363.

[153] F. Fondement, *Concrete syntax definition for modeling languages*, PhD thesis, École Polytechnique Fédérale de Lausanne, Nov. 2007. 10.5075/epfl-thesis-3927.

[154] G. B. Forbach, *Structured system design: Analysis, design, and construction*, Behavior Research Methods & Instrumentation, 13 (1981), pp. 163–171. 10.3758/BF03207927.

[155] M. Fowler, *Microservices*. Online, Mar. 2014. Accessed 2021-Nov-05. http://martinfowler.com/articles/microservices.html.

[156] M. Fowler and J. Lewis, *Microservices: Nur ein weiteres Konzeptin der Softwarearchitektur oder mehr?*, Online Themenspecial Innovation in und durch Architekturen 2015, (2015).

[157] M. S. Fox, M. Barbuceanu, M. Gruninger, and J. Lin, *An Organization Ontology for Enterprise Modelling*, in Simulating Organizations: Computational Models of Institutions and Groups, M. Prietula, K. Carley, and L. Gasser, eds., MIT Press, 1997, pp. 131–152.

[158] M. S. Fox and M. Gruninger, *Enterprise Modeling*, AI Magazine, 19 (1998). 10.1609/aimag.v19i3.1399.

[159] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, *Model-driven development using UML 2.0: promises and pitfalls*, Computer, 39 (2006), pp. 59–66. 10.1109/MC.2006.65.

[160] R. B. France and B. Rumpe, *Model-driven Development of Complex Software: A Research Roadmap*, in Future of Software Engineering (FOSE '07), 2007, pp. 37–54. 10.1109/FOSE.2007.14.

[161] U. Frank, *The MEMO Meta-Metamodel*, tech. rep., Universität Koblenz-Landau - Institut für Wirtshaftsinformatik, June 1998.

[162] ——, *Multi-perspective enterprise modeling (MEMO) conceptual framework and modeling languages*, in Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002, pp. 1258–1267. 10.1109/HICSS.2002.993989.

[163] ——, *MEMO Organisation Modelling Language: (2) Focus on Business Processes*, Tech. Rep. 49, ICB, Dec. 2011.

[164] ——, *The MEMO Meta Modelling Language (MML) and Language Architecture*, Tech. Rep. 43, ICB, Feb. 2011.

[165] ——, *Enterprise Modelling: The Next Steps*, Enterprise Modelling and Information Systems Architectures, 9 (2014), pp. 22–37.

[166] ——, *Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges*, Software and Systems Modeling, 13 (2014), pp. 941–962. 10.1007/s10270-012-0273-9.

[167] U. Frank, P. Maier, and A. C. Bock, *Low code platforms: Promises, concepts andprospects. a comparative study of ten systems*, Tech. Rep. 70, ICB-Research, 2021. 10.17185/duepublico/75244.

[168] D. Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson Education Limited, 2004.

[169] A. Gargantini, E. Riccobene, and P. Scandurra, *A semantic framework for metamodel-based languages*, Automated Software Engineering, 16 (2009), pp. 415–454. 10.1007/s10515-009-0053-0.

[170] Gartner. Online, Oct. 2022. Accessed on 2022-Dec-15. https://www.gartner.com/en/newsroom/press-releases/2022-10-19-gartner-forecasts-worldwide-it-spending-to-grow-5-percent-in-2023.

[171] J. Geskus and J. L. G. Dietz, *Developing Quality Management Systems with DEMO*, in Advances in Enterprise Engineering III. CIAO! EOMAS 2009, A. Albani, J. Barjis, and J. L. G. Dietz, eds., vol. 34 of Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg, 2009, pp. 130–142. 10.1007/978-3-642-01915-9_10.

[172] G. B. Ghantous and A. Gill, *DevOps: Concepts, Practices, Tools, Benefits and Challenges*, in PACIS 2017 Proceedings, 2017. https://aisel.aisnet.org/pacis2017/96.

[173] R. E. Giachetti, *Design of Enterprise Systems: Theory, Architecture, and Methods*, CRC Press, first ed., Jan. 2010.

[174] M. Gogolla and B. Selic, *On teaching descriptive and prescriptive modeling*, in Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS '20), Oct. 2020, pp. 1–9. 10.1145/3417990.3418744.

[175] B. Gold, *Approaches to accelerating product and process development*, Journal of Product Innovation Management, 4 (1987), pp. 81–88.

[176] G. Goldkuhl, M. Lind, and U. Seigerroth, *Method Integration as a Learning Process*, in Proceedings of the Fifth International Conference of the British Computer Society Information Systems Methodologies Specialist Group, N. Jayaratna, B. Fitzgerald, T. Wood-Harper, and J.-M. Larrasquet, eds., Springer-Verlag, 1997, pp. 15–26.

[177] D. L. Goodhue, D. Q. Chen, M. C. Boudreau, A. Davis, and J. Cochran, *Addressing Business Agility Challenges with Enterprise Systems*, MIS Quarterly Executive, 8 (2009), pp. 73–88.

[178] D. Gouveia, *EMOTIONS - Essential Model of Organization Operationalized in a Normalized System*, PhD thesis, Universidade da Madeira, 2020. Forthcoming.

[179] D. Gouveia and D. Aveiro, *Two Protocols for DEMO Engines: PSI or Tell&Agree*, in 15th CIAO! Doctoral Consortium Workshop, June 2015.

[180] S. Graf, I. Ober, and I. Ober, *A real-time profile for UML*, International Journal on Software Tools for Technology Transfer, 8 (2006), pp. 113–127. 10.1007/s10009-005-0213-x.

[181] T. Graml, R. Bracht, and M. Spies, *Patterns of Business Rules to Enable Agile Business Processes*, in 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), IEEE, 2007. 10.1109/EDOC.2007.35.

[182] R. Grangel, J. P. Bourey, R. Chalmeta, and M. Bigand, *UML for Enterprise Modelling: basis for a Model-Driven Approach*, in Enterprise Interoperability, G. Doumeingts, J. Müller, G. Morel, and B. Vallespir, eds., Springer, London, 2007, pp. 91–101. 10.1007/978-1-84628-714-5_9.

[183] T. Gray, D. Bork, and M. De Vries, *A New DEMO Modelling Tool that Facilitates Model Transformations*, in Enterprise, Business-Process and Information Systems Modeling, Springer, Heidelberg, Germany, 2020, pp. 359–374. 10.1007/978-3-030-49418-6_25.

[184] D. Greefhorst and E. Proper, *Architecture Principles: The Cornerstones of Enterprise Architecture*, The Enterprise Engineering Series, Springer, 2011. 10.1007/978-3-642-20279-7.

[185] T. R. Gruber, *A translation approach to portable ontology specifications*, Knowledge Acquisition, 5 (1993), pp. 199–220. 10.1006/knac.1993.1008.

[186] N. Guarino, G. Guizzardi, and J. Mylopoulos, *On the Philosophical Foundations of Conceptual Models*, in 29th International Conference on Information Modelling and Knowledge Bases (EJC 2019), vol. 321 of Frontiers in Artificial Intelligence and Applications, 2019. 10.3233/FAIA200002.

[187] N. Guarino, D. Oberle, and S. Staab, *What Is an Ontology?*, in Handbook on Ontologies, S. Staab and R. Studer, eds., International Handbooks on Information Systems, Springer, Berlin, Heidelberg, 2009, pp. 1–17. 10.1007/978-3-540-92673-3_0.

[188] G. Guizzardi, *Ontological foundations for structural conceptual models*, PhD thesis, University of Twente, Oct. 2005.

[189] ——, *On Ontology, Ontologies, Conceptualizations, Modeling Languages, and (Meta)Models*, in Proceedings of the 2007 Conference on Databases and Information Systems IV: Selected Papers from the Seventh International Baltic Conference DB&IS'2006, IOS Press, 2007, pp. 18–39. 10.5555/1565421.1565425.

[190] G. Guizzardi, L. Ferreira Pires, and M. van Sinderen, *An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages*, in MODELS 2005: Model Driven Engineering Languages and Systems, L. Briand and C. Williams, eds., vol. 3713 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 691–705. 10.1007/11557432_51.

[191] J. Habermas, *The theory of communicative action*, Cambridge: Polity Press, 1986.

[192] B. Hailpern and P. Tarr, *Model-driven development: The good, the bad, and the ugly*, IBM Systems Journal, 45 (2006), pp. 451–461.

[193] A. Haj-Bolouri, S. Purao, M. Rossi, and L. Bernhardsson, *Action Design Research in Practice: Lessons and Concerns*, in Proceedings of the European Conference on Information Systems 2018 (ECIS 2018), U. Frank, K. Kautz, and P. M. Bednar, eds., June 2018.

[194] A. Hallerbach, T. Bauer, and M. Reichert, *Capturing Variability in Business Process Models: The Provop Approach*, Journal of Software Maintenance and Evolution: Research and Practice, 22 (2010), pp. 519–546.

[195] T. Halpin, *Fact-Oriented Modeling: Past, Present and Future*, in Conceptual Modelling in Information Systems Engineering, J. Krogstie, A. L. Opdahl, and S. Brinkkemper, eds., Springer, 2007, pp. 19–38. 10.1007/978-3-540-72677-7_2.

[196] M. Hamman, *Evolvagility: Growing an Agile Leadership Culture from the Inside Out*, Agile Leadership Institute, Jan. 2019.

[197] G. Hardjosumarto, *An Enterprise Ontology basedApproach to Service Specification*, Master's thesis, Delft University of Technology, 2008.

[198] D. Harel and B. Rumpe, *Meaningful modeling: What's the semantics of "semantics"?*, Computer, 37 (2004), pp. 64–72. 10.1109/MC.2004.172.

[199] F. Harmsen, *Situational Method Engineering*, PhD thesis, University of Twente, Jan. 1997.

[200] F. Harmsen, S. Brinkkemper, and J. Oei, *Situational Method Engineering for Information System Project Approaches*, in Methods and Associated Tools for the Information Systems Life Cycle, A. A. Verrijn Stuart and T. W. Olle, eds., Sept. 1994, pp. 169–194.

[201] F. Hasić, J. De Smedt, and J. Vanthienen, *Towards Assessing the Theoretical Complexity of the Decision Model and Notation (DMN)*, in CEUR Workshop Proceedings, vol. 1859, June 2017, pp. 64–71.

[202] J. Heering and M. Mernik, *Domain-Specific Languages for Software Engineering*, in Proceedings of the 35th Hawaii International Conference on System Sciences, vol. 9, IEEE, 2002. 10.1109/HICSS.2002.10113.

[203] C. Heitmeyer, S. Shukla, M. Archer, and E. Leonard, *On Model-Based Software Development*, in Perspectives on the Future of Software Engineering, J. Münch and K. Schmid, eds., Springer, 2013, pp. 49–60. 10.1007/978-3-642-37395-4_4.

[204] K. Henderson and A. Salado, *Value and benefits of model-based systems engineering (MBSE): Evidence from the literature*, Systems Engineering, 24 (2020). 10.1002/sys.21566.

[205] B. Henderson-Sellers, C. Gonzalez-Perez, and J. Ralyte, *Comparison of Method Chunks and Method Fragments for Situational Method Engineering*, Proceedings of the Australian Software Engineering Conference, ASWEC, (2008), pp. 479–488. 10.1109/ASWEC.2008.4483237.

[206] B. Henderson-Sellers and J. Ralyte, *Situational Method Engineering: State-of-the-Art Review*, Journal of Universal Computer Science, (2010).

[207] B. Henderson-Sellers, J. Ralyté, P. J. Ågerfalk, and M. Rossi, *Situational Method Engineering*, Springer-Verlag Berlin Heidelberg, 2014. 10.1007/978-3-642-41467-1.

[208] D. Hendriks, *The selection process of model based platforms*, Master's thesis, Radboud Universiteit Nijmegen, July 2017.

[209] M. Henning, *API Design Matters*, Commun. ACM, 52 (2009), pp. 46–56. 10.1145/1506409.1506424.

[210] M. Herselman and A. Botha, *Evaluating an Artifact in Design Science Research Methodology as was implemented in a resource constrained environment*, in Proceedings of the 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists (SAICSIT), R. J. Barnett, L. Cleophas, D. G. Kourie, D. B. le Roux, and B. W. Watson, eds., Association for Computing Machinery, Sept. 2015. 10.1145/2815782.2815806.

[211] A. R. Hevner, *A Three Cycle View of Design Science Research*, Scandinavian Journal of Information Systems, 19 (2007), pp. 87–92.

[212] A. R. Hevner, S. T. March, J. Park, and S. Ram, *Design science in information systems research*, MIS Quarterly, 28 (2004), pp. 75–105.

[213] K. Hinkelmann, A. Gerber, D. Karagiannis, B. Thoenssen, A. van der Merwe, and R. Woitsch, *A new paradigm for the continuous alignment of business and IT: Combining enterprise architecture modelling and enterprise ontology*, Computers in Industry, 79 (2016), pp. 77–86. 10.1016/j.compind.2015.07.009.

[214] S. Holwell, *Themes, Iteration, and Recoverability in Action Research*, in Information Systems Research - Relevant Theory and Informed Practice, B. Kaplan, D. P. Truex, D. Wastell, A. T. Wood-Harper, and J. I. DeGross, eds., vol. 143 of IFIP Advances in Information and Communication Technology, Springer, June 2004. 10.1007/1-4020-8095-6_20.

[215] J. Hoogervorst, *A framework for enterprise engineering*, International Journal of Internet and Enterprise Management, 7 (2011), pp. 5–40. 10.1504/IJIEM.2011.038381.

[216] S. J. B. A. Hoppenbrouwers, H. A. E. Proper, and T. P. van der Weide, *A Fundamental View on the Process of Conceptual Modeling*, in Conceptual Modeling - ER 2005, L. Delcambre, C. Kop, H. C. Mayr, J. Mylopoulos, and O. Pastor, eds., vol. 3716 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 128–143. 10.1007/11568322_9.

[217] W. S. Humphrey, *Winning with Software: An Executive Strategy*, SEI Series in Software Engineering, Addison-Wesley Professional, 2001.

[218] P. Huysmans, *On the Feasibility of Normalized Enterprises: Applying Normalized Systems Theory to the High-Level Design of Enterprises.*, PhD thesis, University of Antwerp, 2011.

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

[220] P. Huysmans, K. Ven, and J. Verelst, *Using the DEMO methodology for modeling open source software development processes*, Information and Software Technology, 52 (2010), pp. 656–671. 10.1016/j.infsof.2010.02.002.

[221] IEEE, *Standard Glossary of Software Engineering Terminology*, Tech. Rep. 610.12-1990, The Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017, USA, 1990. 10.1109/IEEESTD.1990.101064.

[222] International Organization for Standardization, *ISO/IEC/IEEE 42010:2022 Software, systems and enterprise - Architecture description.* Online, 2022. Accessed 2022-Dec-08. https://www.iso.org/standard/74393.html.

[223] S. Isaac, T. Bock, and Y. Stoliar, *A methodology for the optimal modularization of building design*, Automation in Construction, 65 (2016), pp. 116–124. 10.1016/j.autcon.2015.12.017.

[224] M. Jacobs, C. Droge, S. K. Vickery, and R. Calantone, *Product and Process Modularity's Effects on Manufacturing Agility and Firm Growth Performance*, Journal of Product Innovation Management, 28 (2011), pp. 123–137. 10.1111/j.1540-5885.2010.00785.x.

[225] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, *Microservices: The Journey So Far and Challenges Ahead*, IEEE Software, 35 (2018), pp. 24–35. 10.1109/MS.2018.2141039.

[226] P. Järvinen, *On boundaries between field experiment, action research and design research*, no. 14 in Reports in Information Sciences, University of Tampere, 2012. https://trepo.tuni.fi/handle/10024/66319.

[227] M. A. Jeusfeld, *Metamodel*, in Encyclopedia of Database Systems, L. Liu and M. T. Özsu, eds., Springer US, Boston, MA, 2009, pp. 1727–1730. 10.1007/978-0-387-39940-9_898.

[228] H. Jipeng and L. Zhihang, *A Constraint and Object Oriented Fifth Generation Programming Language and its Compiler and Runtime System*, ArXiv, (2022). 10.48550/arXiv.2206.01024.

[229] L.-O. Johansson, M. Wärja, H. Kjellin, and S. Carlsson, *Graphical modeling techniques and usefulness in the Model Driven Arcitechture: Which are the criteria for a ' good' Computer independent model?*, in Proceedings of The 31st Information Systems Research Seminar in Scandinavia (IRIS31), Jan. 2008.

[230] A. Josey, *Open Agile Architecture - A Standard of the Open Group*, Van Haren Publishing, 2020.

[231] Y. Jung and A. Anttila, *How to look beyond what users say that they want*, in CHI 2007 Extended Abstracts, B. Begole, S. Payne, E. Churchill, R. St. Amant, D. Gilmore, and M. B. Rosson, eds., CHI EA '07, New York, NY, USA, Apr. 2007, Association for Computing Machinery, pp. 1759–1764. Conference on Human Factors in Computing Systems, San Jose, California, USA, April 28–May 3, 2007. 10.1145/1240866.1240896.

[232] J. M. Juran, *Juran's Quality Control Handbook*, McGraw-Hill, 1988.

[233] J. Juviler, *Microservices vs. APIs: What's the Difference?* Online, Nov. 2020. Accessed 2021-Nov-18. https://blog.hubspot.com/website/microservices-vs-api [cited 18-11-2021].

[234] N. Kahani, M. Bagherzadeh, J. R. Cordy, J. Dingel, and D. Varró, *Survey and classification of model transformation tools*, Software & Systems Modeling, 18 (2019), pp. 2361–2397. 10.1007/s10270-018-0665-6.

[235] A. Kanane, *Challenges related to the adoption of Scrum*, Master's thesis, Umeå Universitet, 2014.

[236] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, *Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages*, in MODELS 2006: Model Driven Engineering Languages and Systems, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds., Berlin, Heidelberg, 2006, Springer Berlin Heidelberg, pp. 528–542. 10.1007/11880240_37.

[237] D. Karagiannis and H. Kühn, *Metamodeling Platforms*, in Proceedings of the Third international EC-Web 2002, Springer, 2002.

[238] F. Karlsson and K. Wistrand, *Method Components - Extending the Vision*, in Proceedings of the the 26th Information Systems Research Seminar inScandinavia (IRIS 26), 2003.

[239] I. J. Kashiwagi, *Complexity is in the Eye of the Beholder*, PhD thesis, Delft University of Technology, 2019. 10.4233/uuid:9b434ac4-ebcd-4e23-812d-354d836fdcb3.

[240] S. Kemmis and R. McTaggart, *Participatory Action Research: Communicative Action and the Public Sphere*, in The Sage handbook of qualitative research, N. K. Denzin and Y. S. Lincoln, eds., Sage Publications Ltd., 2005, pp. 559–603.

[241] S. Kent, *Model Driven Engineering*, in Integrated Formal Methods. IFM 2002, M. Butler, L. Petre, and K. Sere, eds., vol. 2335 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2002. 10.1007/3-540-47884-1_16.

[242] S. Kent, A. Evans, and B. Rumpe, *UML Semantics FAQ*, in ECOOP'99 Workshops, A. Moreira and S. Demeyer, eds., vol. 1743 of LNCS, Springer, 1999, pp. 33–56.

[243] R. Keswani, S. Joshi, and A. Jatain, *Software Reuse in Practice*, in 2014 Fourth International Conference on Advanced Computing & Communication Technologies, 2014, pp. 159–162. 10.1109/ACCT.2014.57.

[244] M. Khalfallah, N. Figay, P. Ghodous, and C. Ferreira da Silva, *Cross-organizational Business Processes Modeling Using Design-by-Contract Approach*, in IWEI 2013: Enterprise Interoperability, M. van Sinderen, P. Oude Luttighuis, E. Folmer, and S. Bosems, eds., vol. 133 of LNBIP, Springer, 2013, pp. 77–90. 10.1007/978-3-642-36796-0_8.

[245] M. Khalifa and J. M. Verner, *Drivers for software development method usage*, IEEE Transactions on Engineering Management, 47 (2000), pp. 360–369. 10.1109/17.865904.

[246] S. Khoshafian, *Service Oriented Enterprises*, Auerbach Publications, 2006. 10.1201/9781420013269.

[247] A. Khurana and S. R. Rosenthal, *Towards Holistic "Front Ends" In New Product Development*, Journal of Product Innovation Management, 15 (1998), pp. 57–74. 10.1111/1540-5885.1510057.

[248] D. Kleidermacher and M. Kleidermacher, *Embedded Systems Security - Practical Methods for Safe and Secure Software and Systems Development*, Newnes, 2012.

[249] D. Klein, *When to Ignore What Users Say*, Ergonomics in Design: The Quarterly of Human Factors Applications, 14 (2006), pp. 24–26. 10.1177/106480460601400106.

[250] A. G. Kleppe, *A Language Description is More than a Metamodel*, in 4th International Workshop on Software Language Engineering (ATEM 2007), Nashville, United States, Oct. 2007. https://research.utwente.nl/files/134651864/paper18.pdf.

[251] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Pearson, Apr. 2003.

[252] KMar, *KMar Operationele Werkvloer Activiteiten (KOWA): Hoe werkt dat eigenlijk*. KMarMagazine, Online, Oct. 2014. Article in Dutch, Accessed 2023-Feb-06. https://magazines.defensie.nl/kmarmagazine/2014/10/11-hoe-werkt-dat-10.

[253] P. Koen, G. Ajamian, R. Burkart, A. Clamen, J. Davidson, R. D'Amore, C. Elkins, K. Herald, M. Incorvia, A. Johnson, R. Karol, R. Seibert, A. Slavejkov, and K. Wagner, *Providing Clarity and A Common Language to the 'Fuzzy Front End'* , Research-Technology Management, 44 (2001), pp. 46–55. 10.1080/08956308.2001.11671418.

[254] D. S. Kolovos, R. F. Paige, and F. A. C. Polack, *On-demand merging of traceability links with models*, 3rd ECMDA traceability workshop, (2006), pp. 47–55.

[255] T. Kosar, P. E. Martínez López, P. A. Barrientos, and M. Mernik, *A preliminary study on various implementation approaches of domain-specific language*, Information and Software Technology, 50 (2008), pp. 390–405. 10.1016/j.infsof.2007.04.002.

[256] F. Kossak, C. Illibauer, V. Geist, J. Kubovy, C. Natschläger, T. Ziebermayr, T. Kopetzky, B. Freudenthaler, and K.-D. Schewe, *A Rigorous Semantics for BPMN 2.0 Process Diagrams*, Springer International Publishing, Cham, 2014, ch. A Rigorous Semantics for BPMN 2.0 Process Diagrams, pp. 29–152. 10.1007/978-3-319-09931-6_4.

[257] K. Kraft, *Are product and Process Innovations Independent of Each Other?*, Applied Economics, 22 (1990), pp. 1029–1038. 10.1080/00036849000000132.

[258] J. Krogstie, *Evaluating UML using a generic quality framework*, in UML and the Unified Process, L. Favre, ed., 2003, pp. 1–22. 10.4018/978-1-93177-744-5.ch001.

[259] J. Krogstie, P. McBrien, R. Owens, and A. H. Seltveit, *Information systems development using a combination of process and rule based approaches*, in Advanced Information Systems Engineering. CAiSE

1991, R. Andersen, J. A. Bubenko, and A. Sølvberg, eds., vol. 498 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1991. 10.1007/3-540-54059-8_92.

[260] M. R. Krouwel, *Understanding the Essence of Organizations with DEMO-4*. Online, Jan. 2021. https://www.linkedin.com/pulse/understanding-essence-organizations-demo-4-marien-krouwel/.

[261] ——, *Low Code and No Code: Is There a Difference? A practical analysis*. Online, Feb. 2022. https://www.linkedin.com/pulse/low-code-difference-practical-analysis-marien-krouwel/.

[262] M. R. Krouwel and M. Op 't Land, *Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems*, in EEWC 2011: Advances in Enterprise Engineering V, A. Albani, J. L. G. Dietz, and J. Verelst, eds., vol. 79 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2011, pp. 31–45. 10.1007/978-3-642-21058-7_3.

[263] ——, *Using Enterprise Ontology as a basis for Requirements for Cross-Organizationally Usable Applications*, in Proceedings of the 7^{th} Mediterranean Conference on Information Systems 2012 (MCIS2012), A. D. Figueiredo, I. Ramos, and E. Trauth, eds., MCIS Proceedings, University of Minho, Portugal, Sept. 2012, AIS Electronic Library (AISeL). Paper 23. http://aisel.aisnet.org/mcis2012/23.

[264] ——, *Business Driven Micro Service Design - An Enterprise Ontology based approach to API Specifications*, in Advances in Enterprise Engineering XV, D. Aveiro, H. A. Proper, S. Guerreiro, and M. De Vries, eds., vol. 441 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 95–113. 10.1007/978-3-031-11520-2_7.

[265] M. R. Krouwel, M. Op 't Land, and T. Offerman, *Formalizing Organization Implementation*, in EEWC 2016: Advances in Enterprise Engineering X, D. Aveiro, R. Pergl, and D. Gouveia, eds., vol. 252 of Lecture Notes in Business Information Processing, Funchal, Madeira Island, Portugal, 2016, Springer, pp. 3–18. 10.1007/978-3-319-39567-8_1.

[266] M. R. Krouwel, M. Op 't Land, and H. A. Proper, *Generating Low-Code Applications from Enterprise Ontology*, in PoEM 2022: The Practice of Enterprise Modeling, B. S. Barn and K. Sandkuhl, eds., vol. 456 of Lecture Notes in Business Information Processing, Springer Nature Switzerland AG, 2022, pp. 19–32. 10.1007/978-3-031-21488-2_2.

[267] ——, *From Enterprise Models to Low-Code Applications: Mapping DEMO to Mendix, illustrated in the Social Housing domain*, International Journal on Software and Systems Modeling, (2024). Invited submission, currently under review.

[268] M. R. Krouwel, M. Stam, A. Bulat, and R. de Wit, *Mendix and SAP: do you choose or combine?* Online, Aug. 2021. https://www.linkedin.com/pulse/mendix-sap-do-you-choose-combine-marien-krouwel/.

[269] S. Kuhn, *The Software Design Studio: An Exploration*, IEEE Software, 15 (1998), pp. 65–71. 10.1109/52.663788.

[270] T. Kühne, *Matters of (Meta-) Modeling*, Software and Systems Modeling, 5 (2006), pp. 369–385. 10.1007/s10270-006-0017-9.

[271] S. Kujala, *Effective user involvement in product development by improving the analysis of user needs*, Behaviour & Information Technology, 27 (2008), pp. 457–473. 10.1080/01449290601111051.

[272] K. Kumar and R. J. Welke, *Methodology Engineering: A Proposal for Situation-Specific Methodology Construction*, in Challenges and strategies for research in systems development, W. W. Cotterman and J. A. Senn, eds., John Wiley & Sons, Inc., 605 Third Ave. New York, NY, United States, Sept. 1992, pp. 257–269. 10.5555/133549.133574.

[273] T. Kurozumi, *Outline of the Fifth-Generation Project and ICO Activities*, in Technology Transfer in Consortia and Strategic Alliances, D. V. Gibson and R. W. Smilor, eds., vol. 3 of International series on technical innovation and entrepreneurship, Rowman & Littlefield, 1992, ch. 15, pp. 173–192.

[274] A. Kwan, H.-A. Jacobsen, A. Chan, and S. Samoojh, *Microservices in the Modern Software World*, in Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering, CASCON '16, USA, 2016, IBM Corp., pp. 297–299.

[275] A. Laarman and I. Kurtev, *Ontological Metamodeling with Explicit Instantiation*, in SLE 2009: Software Language Engineering, M. van den Brand, D. Gašević, and J. Gray, eds., vol. 5969 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2010, pp. 174–183. 10.1007/978-3-642-12107-4_14.

[276] P. Lago, H. Muccini, and H. van Vliet, *A Scoped Approach to Traceability Management*, Journal of Systems and Software, 82 (2009), pp. 168–182. 10.1016/j.jss.2008.08.026.

[277] M. M. Lankhorst, *Enterprise Architecture at Work: Modelling, Communication and Analysis*, The Enterprise Engineering Series, Springer, fourth ed., 2017. 10.1007/978-3-662-53933-0.

[278] M. M. Lankhorst, H. A. Proper, and H. Jonkers, *The Anatomy of the ArchiMate Language*, International Journal of Information System Modeling and Design, 1 (2010), pp. 1–32. 10.4018/jismd.2010092301.

[279] B. Lantow, K. Sandkuhl, and J. Stirna, *Enterprise Modeling with 4EM: Perspectives and Method*, in Domain-Specific Conceptual Modeling, D. Karagiannis, M. Lee, K. Hinkelmann, and W. Utz, eds., Springer, 2022, pp. 95–120. 10.1007/978-3-030-93547-4_5.

[280] A. S. Lee, *A Scientific Methodology for MIS Case Studies.*, MIS Quarterly, 13 (1989), pp. 33–50. 10.2307/248698.

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

[282] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, *Metrics and Laws of Software Evolution - The Nineties View*, in Proceedings Fourth International Software Metrics Symposium, IEEE, 1997, pp. 20–32. 10.1109/METRIC.1997.637156.

[283] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, *A Survey of DevOps Concepts and Challenges*, ACM Computing Surveys, 1 (2019). 10.1145/3359981.

[284] M. Leppänen, *A Context-Based Enterprise Ontology*, in Business Information Systems (BIS 2007), W. Abramowicz, ed., vol. 4439 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007, pp. 273–286. 10.1007/978-3-540-72035-5_21.

[285] S. W. Liddle, *Model-Driven Software Development*, in Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges, D. W. Embley and B. Thalheim, eds., Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, ch. Model-Driven Software Development, pp. 17–54. 10.1007/978-3-642-15865-0_2.

[286] J. Liebenau and J. Backhouse, *Understanding Information: An Introduction*, no. 1 in Information Systems Series, Red Globe Press London, 1990. 10.1007/978-1-349-11948-6.

[287] M. Lind and G. Goldkuhl, *Designing Business Process Variants - Using the BAT Framework as a Pragmatic Lens.*, in Business Process Management Workshops, C. Bussler and A. Haller, eds., vol. 3812, Feb. 2005, pp. 408–420.

[288] Linux Foundation, *OpenAPI Specification v3.1.0*. Online, Feb. 2021. Accessed 2021-Nov-21. https://spec.openapis.org/oas/v3.1.0.

[289] M. Loukides and S. Swoyer, *Microservices Adoption in 2020*, tech. rep., O'Reilly, July 2020. Accessed 2022-Jun-01. https://www.oreilly.com/radar/microservices-adoption-in-2020/.

[290] P. Lucena, A. Braz, A. Chicoria, and L. Tizzei, *IBM Design Thinking Software Development Framework*, in Agile Methods, T. Silva da Silva, B. Estácio, J. Kroll, and R. Mantovani Fontana, eds., Cham, 2017, Springer International Publishing, pp. 98–109.

[291] J. Ludewig, *Models in software engineering – an introduction*, Software and Systems Modeling, 2 (2003), pp. 5–14. 10.1007/s10270-003-0020-3.

[292] R. Luitwieler, *A Selection Method for Model-Driven Development Tools*, Master's thesis, TU Delft, 2010.

[293] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, *Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective*, CoRR, (2021). 10.48550/arXiv.2107.07482.

[294] G. Madhavan, *Applied Minds: How Engineers Think*, W. W. Norton & Company, Aug. 2015.

[295] A. Maes and G. Poels, *Evaluating quality of conceptual modelling scripts based on user perceptions*, Data & Knowledge Engineering, 63 (2007), pp. 701–724.

[296] E. B. Magrab, S. K. Gupta, F. P. McCluskey, and P. Sandborn, *Integrated Product and Process Design and Development: The Product Realization Process*, CRC Press, second ed., 2009. 10.1201/9781420070613.

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

[298] H. Mannaert, J. Verelst, and P. De Bruyn, *Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design*, Normalized Systems Institute, Koppa, 2016.

[299] H. Mannaert, J. Verelst, and K. Ven, *Towards evolvable software architectures based on systems theoretic stability*, Software: Practice and Experience, 42 (2012), pp. 89–116. 10.1002/spe.1051.

[300] M. Mannio and U. Nikula, *Requirements Elicitation Using a Combination of Prototypes and Scenarios*, in WER, 2001, pp. 283–296.

[301] S. T. March and G. F. Smith, *Design and Natural Science Research on Information Technology*, Decis. Support Syst., 15 (1995), pp. 251–266. 10.1016/0167-9236(94)00041-2.

[302] A. Marchenko and P. Abrahamsson, *Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*, in Agile 2008 Conference, 2008, pp. 15–26. 10.1109/Agile.2008.77.

[303] C. Marshall, *Enterprise Modeling with UML: Designing Successful Software through Business Analysis*, Addison-Wesley, 2000.

[304] J. Martin, *Fourth-generation languages. Volume I: principles*, Prentice-Hall, Inc., 1985.

[305] ——, *Information Engineering*, vol. 1, Savant, 1986.

[306] L. Mathiassen and A. M. Vainio, *Dynamic Capabilities in Small Software Firms: A Sense-and-Respond Approach*, IEEE Transactions on Engineering Management, 54 (2007), pp. 522–538. 10.1109/TEM.2007.900782.

[307] R. J. Mayer, C. P. Menzel, M. K. Painter, P. S. Dewitte, T. Blinn, and B. Perakath, *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*, tech. rep., Knowledge Based Systems Incorporated, 1995.

[308] M. D. McIlroy, *Mass-Produced Software Components*, in Software Engineering: Report of a conference sponsored by the NATO Science Committee, P. Naur and B. Randell, eds., Scientific Affairs Division, NATO, Jan. 1969, pp. 138–155.

[309] S. M. McMenamin and J. F. Palmer, *Essential Systems Analysis*, Yourdon, 1984.

[310] L. C. Megginson, *Lessons from Europe for American Business*, The Southwestern Social Science Quarterly, 44 (1963), pp. 3–13. http://www.jstor.org/stable/42866937.

[311] S. J. Mellor, *Adapting agile approaches to your project needs*, IEEE Software, 22 (2005), pp. 17–20.

[312] S. J. Mellor and M. Balcer, *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley, 2002.

[313] S. J. Mellor, A. N. Clark, and T. Futagami, *Model-driven development - Guest editor's introduction*, IEEE Software, 20 (2003), pp. 14–18. 10.1109/MS.2003.1231145.

[314] J. Mendling and W. van der Aalst, *Towards EPC Semantics based on State and Context*, in Proceedings of EPK' 06, M. Nüttgens, F. J. Rump, and J. Mendling, eds., 2006, pp. 25–48.

[315] T. Mens and P. Van Gorp, *A Taxonomy of Model Transformation*, Electronic Notes in Theoretical Computer Science, 152 (2006). 10.1016/j.entcs.2005.10.021.

[316] M. Mernik, J. Heering, and A. Sloane, *When and How to Develop Domain-Specific Languages*, ACM Computing Surveys, 37 (2005), pp. 316–344. 10.1145/1118890.1118892.

[317] S. Molly, *Exploring Organizational Implementation Fundamentals in a real enterprise*, Master's thesis, Antwerp Management School, 2014.

[318] B. Morin, N. Harrand, and F. Fleurey, *Model-Based Software Engineering to Tame the IoT Jungle*, IEEE Software, 34 (2017), pp. 30–36. 10.1109/MS.2017.11.

[319] K. Morris, *Infrastructure as Code*, O'Reilly Media, Inc., second ed., 2020.

[320] N. Mouhib, S. Bah, and A. Berrado, *Governing the Generic System Development Process Using Cybernetic Principles*, in 4th World Conference on Complex Systems (WCCS), 2019, pp. 1–6. 10.1109/ICoCS.2019.8930755.

[321] O. Mráz, P. Náplava, R. Pergl, and M. Skotnica, *Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method*, in Advances in Enterprise Engineering XI: EEWC 2017, D. Aveiro, R. Pergl, G. Guizzardi, J. Almeida, R. Magalhães, and H. Lekkerkerk, eds., vol. 284 of Lecture Notes in Business Information Processing, Springer Cham, 2017, pp. 85–98. 10.1007/978-3-319-57955-9_7.

[322] M. Muehlena and M. Indulska, *Modeling languages for business processes and business rules: A representational analysis*, Information Systems, 35 (2010), pp. 379–390. 10.1016/j.is.2009.02.006.

[323] K. S. Mukasa and H. Kaindl, *An Integration of Requirements and User Interface Specifications*, in 2008 16th IEEE International Requirements Engineering Conference, 2008, pp. 327–328. 10.1109/RE.2008.55.

[324] J. B. F. Mulder, *Rapid Enterprise Design.*, PhD thesis, Delft University of Technology, 2006.

[325] M. A. T. Mulder, *Enabling the automatic verification and exchange of DEMO models*, PhD thesis, Radboud University Nijmegen, 2022. https://repository.ubn.ru.nl/handle/2066/247698.

[326] N. Mustafa and Y. Labiche, *The Need for Traceability in Heterogeneous Systems: A Systematic Literature Review*, in IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol. 1, 2017, pp. 305–310. 10.1109/COMPSAC.2017.237.

[327] R. N. Nagel and R. Dove, *Twenty-First CenturyManufacturing Enterprise Strategy: An Industry-LedView*, tech. rep., Iacocca Institute, Lehigh University, 1991.

[328] S. Newman, *Building Microservices*, O'Reilly Media, Inc., 2015.

[329] E. Niemi, *Enterprise Architecture Benefits: Perceptions from Literature and Practice*, in Proceedings of the 7th IBIMA Conference Internet & Information Systems in the Digital Age, 2006.

[330] D. Norman, *The Design of Everyday Things*, Basic Books, revised and expanded ed., 2013.

[331] A. Noureen, A. Amjad, and F. Azam, *Model Driven Architecture - Issues, Challenges and Future Directions*, Journal of Software, 11 (2016), pp. 924–933.

[332] B. Nuseibeh and S. Easterbrook, *Requirements engineering: a roadmap*, in Proceedings of the Conference on The Future of Software Engineering, ICSE '00, New York, NY, USA, 2000, ACM, pp. 35–46. 10.1145/336512.336523.

[333] OASIS, *Web Services Business Process Execution Language Version 2.0*, tech. rep., OASIS, Apr. 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[334] Object Management Group, *Business Motivation Model*, tech. rep., Object Management Group, Aug. 2008. Version 1.0. http://www.omg.org/spec/BMM/1.0/PDF/.

[335] ——, *Model Driven Architecture (MDA) Guide*, tech. rep., Object Management Group, June 2014. Rev. 2.0. https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.

[336] T. Offerman, *Improving IT Supported Organizational Change; Formalizing Organizational Implementation Fundamentals*, Master's thesis, Universiteit Leiden, 2014. https://theses.liacs.nl/pdf/Offerman-Tyron-non-confidential.pdf.

[337] I. Oliver, *Applying UML and MDA to real systems design*, in Design, Automation and Test in Europe, vol. 1, 2005, pp. 70–71. 10.1109/DATE.2005.65.

[338] M. Op 't Land, *Toward Evidence Based Splitting of Organizations*, in Proceedings of the IFIP WG 8.1 Working Conference on Situational Method Engineering (ME07), Fundamentals and Experiences, J. Ralyté, S. Brinkkemper, and B. Henderson-Sellers, eds., vol. IFIP 244 of IFIP Series, Geneva, Switzerland, Sept. 2007, Springer-Verlag Berlin Heidelberg 2007, pp. 328–342.

[339] ——, *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*, PhD thesis, Delft University of Technology, June 2008. http://resolver.tudelft.nl/uuid:0edd0472-39df-4296-b692-e9916e79fb1e.

[340] M. Op 't Land and J. L. G. Dietz, *Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations*, in Proceedings of the 2^{nd} Enterprise Engineering Working Conference (EEWC-2012), A. Albani, D. Aveiro, and J. Barjis, eds., LNBIP 110, Springer, Heidelberg, May 2012, pp. 77–92. http://resolver.tudelft.nl/uuid:18d90cf6-fddc-48ae-8101-e9eda186a72c.

[341] M. Op 't Land and M. R. Krouwel, *Exploring Organizational Implementation Fundamentals*, in EEWC 2013: Advances in Enterprise Engineering VII, H. A. Proper, D. Aveiro, and K. Gaaloul, eds., vol. 146 of Lecture Notes in Business Information Processing, Springer-Verlag Berlin Heidelberg, 2013, pp. 28–42. 10.1007/978-3-642-38117-1_3.

[342] M. Op 't Land, M. R. Krouwel, and S. Gort, *Testing the Concept of the RUN-Time Adaptive Enterprise - Combining Organization and IT Agnostic Enterprise Models with Organization Implementation Variables and Low Code Technology*, in EEWC 2020: Advances in Enterprise Engineering XIV, D. Aveiro, G. Guizzardi, R. Pergl, and H. A. Proper, eds., vol. 411 of Lecture Notes in Business Information Processing, Springer, Apr. 2021, pp. 228–242. 10.1007/978-3-030-74196-9_13.

[343] M. Op 't Land, M. R. Krouwel, E. van Dipten, and J. Verelst, *Exploring Normalized Systems Potential for Dutch MoD's Agility – A Proof of Concept on Flexibility, Time-to-Market, Productivity and Quality*, in PRET 2011: Practice-Driven Research on Enterprise Transformation, F. Harmsen, K. Grahlmann, and E. Proper, eds., vol. 89 of Lecture Notes in Business Information Processing, Springer, Sept. 2011, pp. 110–121. 10.1007/978-3-642-23388-3_5.

[344] M. Op 't Land and J. Pombinho, *Strengthening the Foundations Underlying the Enterprise Engineering Manifesto*, in Proceedings of the 2nd Enterprise Engineering Working Conference (EEWC-2012), A. Albani, D. Aveiro, and J. Barjis, eds., LNBIP 110, Springer, Heidelberg, May 2012, pp. 1–14. http://resolver.tudelft.nl/uuid:6d36c4d9-5489-44bb-8223-a7f4bd7693cf.

[345] M. Op 't Land and H. A. Proper, *Impact of Principles on Enterprise Engineering.*, in Proceedings of the 15th European Conference on Information Systems, H. Österle, J. Schelp, and R. Winter, eds., St. Gallen, Switzerland, June 2007, University of St. Gallen, pp. 1965–1976. http://resolver.tudelft.nl/uuid:577a88b0-9b5a-49f4-94ec-0e7dbb00c1ab.

[346] M. Op 't Land, H. Zwitzer, P. Ensink, and Q. Lebel, *Towards a Fast Enterprise Ontology Based Method for Post Merger Integration*, in Proceedings of the 2009 ACM symposium on Applied Computing (SAC-ACM2009), SAC '09, New York, NY, USA, Mar. 2009, ACM, pp. 245–252. 10.1145/1529282.1529336.

[347] D. L. Orlovskyi and A. M. Kopp, *Towards the business process model as code approach*, in Proceedings of the International Science-tech Conference, 2020, pp. 6–9. http://repository.kpi.kharkov.ua/handle/KhPI-Press/50038.

[348] E. Overby, A. Bharadwaj, and V. Sambamurthy, *Enterprise agility and the enabling role of information technology*, Eur. J. Inf. Syst., 15 (2006), pp. 120–131. 10.1057/palgrave.ejis.3000600.

[349] D. Pacheco, D. Aveiro, B. Gouveia, and D. Pinto, *Evaluation of the Perceived Quality and Functionality of Fact Model Diagrams in DEMO*, in Advances in Enterprise Engineering XV, D. Aveiro, H. A. Proper, S. Guerreiro, and M. de Vries, eds., vol. 441 of Lecture Notes in Business Informa-

tion Processing, Springer, 2022, pp. 114–128. 10.1007/978-3-031-11520-2_8.

[350] D. Pacheco, D. Aveiro, D. Pinto, and B. Gouveia, *Towards the X-Theory: An Evaluation of the Perceived Quality and Functionality of DEMO's Process Model*, in Advances in Enterprise Engineering XV, D. Aveiro, H. A. Proper, S. Guerreiro, and M. de Vries, eds., vol. 441 of Lecture Notes in Business Information Processing, Springer, 2022, pp. 129–148. 10.1007/978-3-031-11520-2_9.

[351] J. Pacheco, S. Garbatov, and M. Goulão, *Improving Collaboration Efficiency Between UX/UI Designers and Developers in a Low-Code Platform*, in 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 138–147. 10.1109/MODELS-C53483.2021.00025.

[352] P. Parviainen, J. Takalo, S. Teppola, and M. Tihinen, *Model-Driven Development: Processes and practices*, no. 114 in VTT Working Papers, VTT Technical Research Centre of Finland, Feb. 2009.

[353] C. Páscoa, P. Sousa, and J. Tribolet, *Ontology Construction: Representing Dietz "Process" and "State" Models Using BPMN Diagrams*, in Enterprise Information Systems Design, Implementation and Management: Organizational Applications, M. M. Cruz-Cunha and J. Varajao, eds., Information Science Reference, July 2010. 10.4018/978-1-61692-020-3.ch004.

[354] N. Passos and J. L. Pereira, *Business Process Modeling: how CMMN and DMN complement BPMN*, in CAPSI 2018 Proceedings, vol. 7, 2018.

[355] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, *A Design Science Research Methodology for Information Systems Research*, Journal of Management Information Systems, 24 (2007), pp. 45–77. 10.2753/MIS0742-1222240302.

[356] A. M. Petersson and J. Lundberg, *Applying Action Design Research (ADR) to Develop Concept Generation and Selection Methods*, Procedia CIRP, 50 (2016), pp. 222–227. 10.1016/j.procir.2016.05.024.

[357] M. Pidd, *Tools for Thinking - Modeling in Management Science*, Wiley, 1997.

[358] D. Pinto, D. Aveiro, D. Pacheco, B. Gouveia, and D. Gouveia, *Fact Model in DEMO - Urban Law Case and Proposal of Representation Improvements*, in Advances in Enterprise Engineering XIV, D. Aveiro, G. Guizzardi, R. Pergl, and H. A. Proper, eds., vol. 411 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 173–190. 10.1007/978-3-030-74196-9_10.

[359] ——, *Validation of DEMO's Conciseness Quality and Proposal of Improvements to the Process Model*, in Advances in Enterprise Engineering XIV, D. Aveiro, G. Guizzardi, R. Pergl, and H. A. Proper, eds., vol. 411 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 133–152. 10.1007/978-3-030-74196-9_8.

[360] K. Pohl, *Process-Centered Requirements Engineering*, John Wiley Research Science Press, 1996.

[361] B. Porter and R. R. Filho, *Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning*, in 2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2016, pp. 40–49. 10.1109/SASO.2016.10.

[362] J. Prieler, T. Aicher, and B. Vogel-Heuser, *Increasing adaptability of automation control software for automated material flow systems via software modularization*, in IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, 2017, pp. 3951–3956. 10.1109/IECON.2017.8216676.

[363] H. A. Proper and D. Greefhorst, *Principles in an Enterprise Architecture Context*, Journal of Enterprise Architecture, (2011).

[364] H. A. Proper and M. Op 't Land, *Lines in the Water – The Line of Reasoning in an Enterprise Engineering Case Study from the Public Sector*, in Working Conference on Practice-Driven Research on Enterprise Transformation, Springer, 2010, pp. 193–216.

[365] P. Rai and S. Dhir, *Impact of Different Methodologies in Software Development Process*, International Journal of Computer Science and Information Technologies, 5 (2014), pp. 1112–1116.

[366] A. Raj, P. Tadinada, and S. Hendryx, *Transformation of SBVR business design to UML models*, in Proceeding of the 1st Annual India Software Engineering Conference (ISEC 2008), Feb. 2008, pp. 29–38. 10.1145/1342211.1342221.

[367] J. Ralyté, R. Deneckère, and C. Rolland, *Towards a Generic Model for Situational Method Engineering*, in Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), J. Eder and M. Missikoff, eds., vol. 2681 of Lecture Notes in Computer Science, Springer, June 2003, pp. 95–110. 10.1007/3-540-45017-3_9.

[368] J. Ralyté and C. Rolland, *An Assembly Process Model for Method Engineering*, in Advanced Information Systems Engineering, K. R. Dittrich, A. Geppert, and M. C. Norrie, eds., Berlin, Heidelberg, June 2001, Springer Berlin Heidelberg, pp. 267–283. 10.1007/3-540-45341-5_18.

[369] B. Ramesh, C. Stubbs, T. Powers, and M. Edwards, *Requirements traceability: Theory and practice*, Annals of Software Engineering, 3 (1997), pp. 397–415. 10.1023/A:1018969401055.

[370] J. Recker, M. Indulska, M. Rosemann, and P. Green, *How Good Is BPMN Really? Insights from Theory and Practice*, in Proceedings of the 14th European Conference on Information Systems, J. Ljungberg and M. Andersson, eds., Goeteborg, Sweden, 2006, pp. 1582–1593.

[371] G. Regev, P. Soffer, and R. Schmidt, *Taxonomy of Flexibility in Business Processes*, in BPMDS, G. Regev, P. Soffer, and R. Schmidt, eds., vol. 236 of CEUR Workshop Proceedings, CEUR-WS.org, June 2006.

[372] K. Reilly, *What is low code development.* Online, Aug. 2019. Accessed 2020-Aug-31. https://medium.com/swlh/what-is-low-code-development-f45550c3243d.

[373] D. G. Reinertsen, *Taking the Fuzziness Out of the Fuzzy Front End*, Research-Technology Management, 42 (1999), pp. 25–31. 10.1080/08956308.1999.11671314.

[374] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano, *On the effectiveness of screen mockups in requirements engineering: Results from an internal replication*, in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2010), G. Succi, M. Morisio, and N. Nagappan, eds., ACM, Sept. 2010. 10.1145/1852786.1852809.

[375] C. Richardson and J. R. Rymer, *New Development Platforms Emerge For Customer-Facing Applications*, tech. rep., Forrester, 2014.

[376] ——, *Vendor Landscape: The Fractured, Fertile Terrain Of Low-Code Application Platforms*, tech. rep., Forrester, Jan. 2016.

[377] J. Rivero, G. Rossi, J. Grigera, E. Robles Luna, and A. Navarro, *From Interface Mockups to Web Application Models*, in Web Information System Engineering - WISE 2011, A. Bouguettaya, M. Hauswirth, and L. Liu, eds., vol. 6997 of Lecture Notes in Computer Science, Jan. 2011, pp. 257–264. 10.1007/978-3-642-24434-6_20.

[378] J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, F. Montero, and M. Gaedke, *Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering*, Information and Software Technology, 56 (2014), pp. 670–687. 10.1016/j.infsof.2014.01.011.

[379] N. Roberts and V. Grover, *Investigating firm's customer agility and firm performance: The importance of aligning sense and respond capabilities*, Journal of Business Research, 65 (2012), pp. 579–585. 10.1016/j.jbusres.2011.02.009.

[380] A. Rochfeld and H. Tardieu, *MERISE: An information system design and development methodology*, Information & Management, 6 (1983), pp. 143–159. 10.1016/0378-7206(83)90032-0.

[381] K. Rokis and M. Kirikova, *Challenges of Low-Code/No-Code Software Development: A Literature Review*, in Perspectives in Business Informatics Research, Ē. Nazaruka, K. Sandkuhl, and U. Seigerroth, eds., vol. 462 of Lecture Notes in Business Information Processing, Springer International Publishing, 2022, pp. 3–17. 10.1007/978-3-031-16947-2_1.

[382] C. Rolland, V. Plihon, and J. Ralyté, *Specifying the reuse context of scenario method chunks*, in Advanced Information Systems Engineering. CAiSE 1998, B. Pernici and C. Thanos, eds., vol. 1413 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, June 1998, pp. 191–218. 10.1007/BFb0054226.

[383] F. Rosique, P. Sánchez, D. Alonso, and M. Jiménez-Buendía, *Traceability Support for MDE Development of Home Automation Systems*, in Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT), SciTePress, July 2012, pp. 224–229. 10.5220/0004081302240229.

[384] D. Ross and K. Schoman, *Structured Analysis for Requirements Definition*, IEEE Transactions on Software Engineering, 3 (1977).

[385] R. G. Ross, *Principles of the Business Rule Approach*, Information Technology Series, Addison-Wesley Professional, 2003.

[386] ——, *The Business Rule Approach*, IT SYSTEMS PERSPECTIVE, (2003).

[387] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, *The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development*, in Proceedings of the 28th International Conference on Intelligent User Interfaces, IUI '23, New York, NY, USA, 2023, Association for Computing Machinery, pp. 491–514. 10.1145/3581641.3584037.

[388] D. C. F. Rothengatter, *Engineering situational methods for professional service organizations: An action design research approach*, PhD thesis, University of Twente, 2012.

[389] E. Rovida and M. Fargnoli, *Some Considerations about Design Education*, in International Design Conference - DESIGN 2004, May 2004.

[390] N. B. Ruparelia, *Software Development Lifecycle Models*, ACM SIGSOFT Software Engineering Notes, 35 (2010), pp. 8–13. 10.1145/1764810.1764814.

[391] I. Rychkova, *Towards Automated Support for Case Management Processes with Declarative Configurable Specifications*, in Business Process Management Workshops, M. Rosa and P. Soffer, eds., vol. 132 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2013, pp. 65–76. 10.1007/978-3-642-36285-9_9.

[392] V. Sambamurthy, A. S. Bharadwaj, and V. Grover, *Shaping Agility Through Digital Options: Reconceptualizing the Role of Information Technology in Contemporary Firms*, MIS Quarterly, 27 (2003), pp. 237–263. 10.2307/30036530.

[393] R. Sanchis, Ó. García-Perales, F. Fraile, and P. Poler, *Low-Code as Enabler of Digital Transformation in Manufacturing Industry*, Applied Sciences, 10 (2019), p. 12. 10.3390/app10010012.

[394] K. Sandkuhl, J. Stirna, A. Persson, and M. Wissotzki, *Enterprise Modeling: Tackling Business Challenges with the 4EM Method*, Springer, Sept. 2014. 10.1007/978-3-662-43725-4.

[395] K. Sandoval, *Ultimate Guide To 9 Common HTTP Methods*. Online, Jan. 2020. Accessed 2022-Jan-04. https://nordicapis.com/ultimate-guide-to-all-9-standard-http-methods/.

[396] I. Santiago, Á. Jiménez, J. M. Vara, V. de Castro, V. A. Bollati, and E. Marcos, *Model-Driven Engineering as a new landscape for traceability management: A systematic literature review*, Information and Software Technology, 54 (2012), pp. 1340–1356. 10.1016/j.infsof.2012.07.008.

[397] J. Sarkis, *Benchmarking for agility*, Benchmarking: An International Journal, 8 (2001), pp. 88–107. 10.1108/14635770110389816.

[398] S. Sattari Khavas, *The adoption of DEMO in practice*, Master's thesis, TU Delft, 2010. http://repository.tudelft.nl/view/ir/uuid%3A7bf74f2c-0cd3-4eb8-b9eb-2f931cae3694/.

[399] A. Schatten and J. Schiefer, *Agile Business Process Management with Sense and Respond*, in IEEE International Conference on e-Business Engineering (ICEBE'07), 2007, pp. 319–322. 10.1109/ICEBE.2007.43.

[400] A.-W. Scheer and M. Nüttgens, *ARIS Architecture and Reference Models for Business Process Management*, in Business Process Management, Models, Techniques, and Empirical Studies (BPM2000), W. M. P. van der Aalst, J. Desel, and A. Oberweis, eds., vol. 1806 of Lecture Notes in Computer Science, Springer, Jan. 2000, pp. 376–389. 10.1007/3-540-45594-9_24.

[401] A.-W. Scheer, O. Thomas, and O. Adam, *Process Modeling using Event-Driven Process Chains*, in Process-Aware Information Systems: Bridging People and Software through Process Technology, M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, eds., John Wiley & Sons, Inc., 2005, pp. 119–145. 10.1002/0471741442.ch6.

[402] G. Scheithauer and S. Hellmann, *Analysis and Documentation of Knowledge-Intensive Processes*, in Business Process Management Workshops, M. Rosa and P. Soffer, eds., vol. 132 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2013, pp. 3–11. 10.1007/978-3-642-36285-9_2.

[403] D. C. Schmidt, *Guest Editor's Introduction: Model-Driven Engineering*, Computer, 39 (2006), pp. 25–31. 10.1109/MC.2006.58.

[404] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst, *Towards a Taxonomy of Process Flexibility*, in CAiSE Forum, Z. Bellahsène, C. Woo, E. Hunt, X. Franch, and R. Coletta, eds., vol. 344 of CEUR Workshop Proceedings, CEUR-WS.org, June 2008, pp. 81–84.

[405] C. Schroth, *The service-oriented enterprise*, Journal of Enterprise Architecture, 3 (2007), pp. 73–80.

[406] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, Cambridge, London, 1969.

[407] G. Sebastián, J. A. Gallud, and R. Tesoriero, *Code generation using model driven architecture: A systematic mapping study*, Journal of Computer Languages, 56 (2020). 10.1016/j.cola.2019.100935.

[408] E. Seidewitz, *What models mean*, IEEE Software, 20 (2003), pp. 26–32. 10.1109/MS.2003.1231147.

[409] M. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, *Action Design Research*, MIS Quarterly, 35 (2011), pp. 37–56. 10.2307/23043488.

[410] B. Selic, *The pragmatics of model-driven development*, IEEE Software, 20 (2003), pp. 19–25. 10.1109/MS.2003.1231146.

[411] P. S. Seligman, G. M. Wijers, and H. G. Sol, *Analyzing the structure of I.S. methodologies; an alternative approach.*, Proceedings of the First Dutch Conference on Information Systems, (1989).

[412] S. Sendall and W. Kozaczynski, *Model transformation: the heart and soul of model-driven software development*, IEEE Software, 20 (2003), pp. 42–45. 10.1109/MS.2003.1231150.

[413] W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Research for Generalized Causal Inference*, Houghton Mifflin, Boston, MA, second ed., 2002.

[414] B. Sherehiy, W. Karwowski, and J. K. Layer, *A review of enterprise agility: Concepts, frameworks, and attributes*, International Journal of Industrial Ergonomics, 37 (2007), pp. 445–460. 10.1016/j.ergon.2007.01.007.

[415] A. Shutov, Y. Laryushina, N. Ponomarev, and O. Radzinskaia, *Extending DEMO with an opportunity for simulation*, in Information Systems Development and Applications, S. Wrycza, ed., University of Gdańsk, 2015.

[416] J. Sijtstra, *Quantifying low-code development platforms effectiveness in the Dutch public sector*, mathesis, Leiden University, June 2022. https://theses.liacs.nl/2221.

[417] H. A. Simon, *The shape of automation for men and management*, Harper & Row, 1965.

[418] ——, *The Sciences of the Artificial*, MIT Press, Cambridge, MA, USA, third ed., 1996.

[419] Y. Singh and M. Sood, *Model Driven Architecture: A Perspective*, in 2009 IEEE International Advance Computing Conference, 2009, pp. 1644–1652. 10.1109/IADCC.2009.4809264.

[420] A. Skander, L. Roucoules, and J.-S. Klein Meyer, *Design and manufacturing interface modelling formanufacturing processes selection and knowledgesynthesis in design*, The International Journal of Advanced Manufacturing Technology, 37 (2008), pp. 443–454. 10.1007/s00170-007-1003-2.

[421] M. Skotnica, S. J. H. van Kervel, and R. Pergl, *Towards the Ontological Foundations for the Software Executable DEMO Action and Fact Models*, in Advances in Enterprise Engineering X (EEWC 2016), D. Aveiro, R. Pergl, and D. Gouveia, eds., vol. 252 of Lecture Notes in Business Information Processing, Springer, 2016, pp. 151–165. 10.1007/978-3-319-39567-810.

[422] J. Smeds, K. Nybom, and I. Porres, *DevOps: A Definition and Perceived Adoption Impediments*, in XP 2015: Agile Processes in Software Engineering and Extreme Programming, C. Lassenius, T. Dingsøyr, and M. Paasivaara, eds., vol. 212 of Lecture Notes in Business Information Processing, Springer, 2015, pp. 166–177. 10.1007/978-3-319-18612-2_14.

[423] R. P. Smith and J. A. Morrow, *Product development process modeling*, Design Studies, 20 (1999), pp. 237–261. 10.1016/S0142-694X(98)00018-0.

[424] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, *The pains and gains of microservices: A Systematic grey literature review*, Journal of Systems and Software, 146 (2018), pp. 215–232. 10.1016/j.jss.2018.09.082.

[425] M. Söylemez, B. Tekinerdogan, and A. Kolukisa Tarhan, *Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review*, Applied Sciences, 12 (2022). 10.3390/app12115507.

[426] G. Spanoudakis and A. Zisman, *Software Traceability: A Roadmap*, Handbook of Software Engineering and Knowledge Engineering, 3 (2005). 10.1142/9789812775245_0014.

[427] P. Spyns, R. Meersman, and M. Jarrar, *Data Modelling versus Ontology Engineering*, SIGMOD Rec., 31 (2002), pp. 12–17. 10.1145/637411.637413.

[428] T. Stahl and M. Völter, *Model-Driven Software Development: Technology, Engineering, Management*, John Wiley & Sons, Inc., Hoboken, NJ, United States, May 2006.

[429] I. Stanev and M. Koleva, *Why the standard methods, 5GL, common platforms and reusable components are the four pillars of the new computational paradigm Programming without programmers*, International Journal of Education and Information Technologies, 13 (2019), pp. 49–58.

[430] L. Starr, A. Mangogna, and S. J. Mellor, *Models to Code - With No Mysterious Gaps*, Apress Berkeley, 2017. 10.1007/978-1-4842-2217-1.

[431] C. Steghuis, *Service granularity in SOA-projects : a trade-off analysis*, Master's thesis, University of Twente, June 2006. http://essay.utwente.nl/57339/.

[432] E. Stevens, *Fuzzy front-end learning strategies: Exploration of a high-tech company*, Technovation, 34 (2014), pp. 431–440. Risk and Uncertainty Management in Technological Innovation. 10.1016/j.technovation.2013.12.006.

[433] V. Stirbu, M. Raatikainen, J. Röntynen, V. Sokolov, T. Lehtonen, and T. Mikkonen, *Towards multi-concern software development with Everything-as-Code*, IEEE Software, (2022).

[434] J. Stirna, A. Person, and K. Sandkuhl, *Participative Enterprise Modeling: Experiences and Recommendations*, in Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, vol. 4495 of Lecture Notes in Computer Science, 2007, pp. 546–560. 10.1007/978-3-540-72988-4_38.

[435] J. Stirna and A. Persson, *Enterprise Modeling: Facilitating the Process and the People*, Springer Cham, Oct. 2018. 10.1007/978-3-319-94857-7.

[436] R. Studer, V. R. Benjamins, and D. Fensel, *Knowledge engineering: Principles and methods*, Data & Knowledge Engineering, 25 (1998), pp. 161–197. 10.1016/S0169-023X(97)00056-6.

[437] S. Sushil and E. A. Stohr, eds., *The Flexible Enterprise*, Flexible Systems Management, Springer India, Jan. 2014. 10.1007/978-81-322-1560-8.

[438] G. I. Susman and R. D. Evered, *An Assessment of the Scientific Merits of Action Research*, Administrative Science Quarterly, 23 (1978), pp. 582–603. 10.2307/2392581.

[439] J. Sztipanovits, *Model-based Software Development*, in ESMD-SW Workshop, Mar. 2007.

[440] T. Tamm, P. B. Seddon, G. Shanks, and P. Reynolds, *How Does Enterprise Architecture Add Value to Organisations?*, Communications of the Association for Information Systems, 28 (2011), pp. 141–168. 10.17705/1CAIS.02810.

[441] D. J. Teece, *Explicating dynamic capabilities: the nature and of (sustainable) enterprise performance*, Strategic Management Journal, 4 (2007), pp. 1319–1350.

[442] S. Teegavarapu, J. D. Summers, and G. M. Mocko, *Case Study Method for Design Research: A Justification*, in International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 2008, pp. 495–503. 10.1115/DETC2008-49980.

[443] A. H. M. ter Hofstede and H. A. Proper, *How to Formalize It? Formalization Principles for Information System Development Methods*, Information and Software Technology, 40 (1998), pp. 519–540.

[444] L. I. Terlouw and A. Albani, *An Enterprise Ontology-Based Approach to Service Specification*, IEEE Transactions on Services Computing, 6 (2013), pp. 89–101. 10.1109/TSC.2011.38.

[445] B. Thalheim and H. Jaakola, *Model-Based Fifth Generation Programming*, in Information Modelling and Knowledge Bases XXXI, A. Dahanayake, J. Huiskonen, Y. Kiyoki, B. Thalheim, H. Jaakola, and N. Yoshida, eds., vol. 321 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2020, pp. 381–400.

[446] B. Thalheim, A. N. Sotnikov, and I. Fiodorov, *Models : The Main Tool of True Fifth Generation Programming*, in Proceedings of the XXII International Conference Enterprise Engineering and Knowledge Management, G. Osipov, Y. Telnov, and I. Fiodorov, eds., vol. 2413 of CEUR Workshop Proceedings, 2019, pp. 161–170. https://ceur-ws.org/Vol-2413/paper19.pdf.

[447] The Open Group, *Microservices Architecture*. Online, 2016. Accessed 2021-Nov-21. http://www.opengroup.org/soa/source-book/msawp/.

[448] ——, *TOGAF – The Open Group Architectural Framework*. Online, 2018. Accessed 2020-Aug-31. http://www.togaf.org.

[449] ——, *ArchiMate 3.2 Specification*, Van Haren, 2023.

[450] O. Thomas and M. Fellmann, *Semantic EPC: Enhancing Process Modeling Using Ontology Languages*, in Semantic Business Processand Product Lifecycle Management. Proceedings of the Workshop SBPM 2007, M. Hepp, K. Hinkelmann, D. Karagiannis, R. Klein, and N. Stojanovic, eds., vol. 251, Jan. 2007.

[451] S. H. Thomke, *The role of flexibility in the development of new products: An empirical study*, Research Policy, 26 (1997), pp. 105–119. 10.1016/S0048-7333(96)00918-3.

[452] Q.-N. N. Tran and G. Low, *MOBMAS: A methodology for ontology-based multi-agent systems development*, Information and Software Technology, 50 (2008), pp. 697–722. 10.1016/j.infsof.2007.07.005.

[453] L. Tratt, *Model transformations and tool integration*, Software and System Modeling, 4 (2005), pp. 112–122. 10.1007/s10270-004-0070-1.

[454] J. Tribolet, *An Engineering Approach to Natural Enterprise Dynamics: From Top-down Purposeful Systemic Steering to Bottom-up Adaptive Guidance Control*, in 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), S. Hammoudi, L. A. Maciaszek, and J. Cordeiro, eds., Apr. 2014.

[455] N. C. Tsourveloudis and K. P. Valavanis, *On the Measurement of Enterprise Agility*, Journal of Intelligent and Robotic Systems, 33 (2002), pp. 329–342. 10.1023/A:1015096909316.

[456] M. Uschold and M. King, *Towards a Methodology for Building Ontologies*, in Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, International Joint Conference on Artificial Intelligence, Montreal, 1995, 1995.

[457] J. E. van Aken and A. P. Nagel, *Organising and managing the fuzzy front end of new product development*, vol. 04.12 of ECIS working paper series, Technische Universiteit Eindhoven, 2004.

[458] S. van Bockhooven and M. Op 't Land, *Organization Implementation Fundamentals: a Case Study Validation in the Youthcare Sector*, in Complementary Proceedings of the Workshops TEE, CoBI, and XOC-BPM at IEEE-COBI 2015, vol. 1408 of CEUR Workshop Proceedings, Lisbon, Portugal, July 2015. http://ceur-ws.org/Vol-1408/paper3-tee.pdf.

[459] H. van den Berg, H. Franken, and H. Jonkers, eds., *Handboek Business Process Engineering*, BiZZdesign Academy B.V., 2008. In Dutch.

[460] W. van der Aalst, J. Desel, and E. Kindler, *On the semantics of EPCs: A vicious circle*, in Proceedings of the EPK 2002: Business Process Management using EPCs, Dec. 2002, pp. 71–80.

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

[462] R. Van Der Straeten, T. Mens, and S. Van Baelen, *Challenges in Model-Driven Software Engineering*, in Models in Software Engineering, M. R. V. Chaudron, ed., Berlin, Heidelberg, 2009, Springer Berlin Heidelberg, pp. 35–47. 10.1007/978-3-642-01648-6_4.

[463] A. van Deursen, P. Klint, and J. Visser, *Domain-Specific Languages: An Annotated Bibliography*, ACM SIGPLAN Notices, 35 (2000), pp. 26–36.

[464] T. van Eijndhoven, M.-E. Iacob, and M. L. Ponisio, *Achieving Business Process Flexibility with Business Rules*, in 12th International IEEE Enterprise Distributed Object Computing Conference, IEEE, 2008, pp. 95–104. 10.1109/EDOC.2008.23.

[465] S. J. H. van Kervel, *Ontology driven Enterprise Information Systems Engineering*, PhD thesis, TU Delft, 2012.

[466] S. J. H. van Kervel, J. L. G. Dietz, J. Hintzen, T. van Meeuwen, and B. Zijlstra, *Enterprise Ontology Driven Software Engineering*, in Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT-2012), 2012, pp. 205–210. 10.5220/0004080902050210.

[467] D. Van Nuffel, *Towards designing modular and evolvable business processes*, PhD thesis, University of Antwerp, 2011.

[468] D. Van Nuffel, P. Huysmans, D. Bellens, and K. Ven, *Translating Ontological Business Transactions into Evolvable Information Systems*, in 5th InternationalConference on Software Engineering Advances (ICSEA 2010), Aug. 2010.

[469] D. Van Nuffel, H. Mulder, and S. van Kervel, *Enhancing the Formal Foundations of BPMN by Enterprise Ontology*, in CIAO! 2009, EOMAS 2009: Advances in Enterprise Engineering III, A. Albani, J. Barjis, and J. L. G. Dietz, eds., vol. 34 of Business Information Processing, Springer, 2009, pp. 115–129. 10.1007/978-3-642-01915-9_9.

[470] M. van Oosterhout, *Business Agility and Information Technology in Service Organizations*, PhD thesis, Erasmus University Rotterdam, June 2010.

[471] M. van Oosterhout, E. Waarts, and J. van Hillegersberg, *Change Factors requiring agility and implications for IT*, European journal of information systems, 15 (2006), pp. 132–145. 10.1057/palgrave.ejis.3000601.

[472] V. E. Van Reijswoud, *The structure of business communication: Theory, model and application*, PhD thesis, Delft University of Technology, 1996.

[473] V. E. van Reijswoud, J. B. F. Mulder, and J. L. G. Dietz, *Communicative Action Based Business Process and Information Modelling with DEMO*, The Information Systems Journal, 9 (1999), pp. 117–138.

[474] J. van't Wout, M. Waage, H. Hartman, M. Stahlecker, and A. Hofman, *The Integrated Architecture Framework Explained: Why, What, How*, Springer, 2010. 10.1007/978-3-642-11518-9.

[475] F. Vernadat, *Enterprise Modelling*, Production Planning & Control, 12 (2001), pp. 107–109. 10.1080/09537280150501202.

[476] P. Verschuren and R. Hartog, *Evaluation in Design-Oriented Research*, Quality and Quantity, 39 (2005), pp. 733–762. 10.1007/s11135-005-3150-6.

[477] C. Vetterli, W. Brenner, F. Uebernickel, and C. Petrie, *From Palaces to Yurts - Why Requirements Engineering Needs Design Thinking*, IEEE Internet Computing, 17 (2013). 10.1109/MIC.2013.32.

[478] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, K. Czarnecki, and B. von Stockfleth, *Model-Driven Software Development: Technology, Engineering, Management*, Wiley Software Patterns Series, Wiley, 2013.

[479] Y. Wand and R. Weber, *Research Commentary: Information Systems and Conceptual Modeling - A Research Agenda*, Information Systems Research, 13 (2002), pp. 363–376.

[480] P. T. Ward and S. J. Mellor, *Structured Development for Real-time Systems: Introduction & tools*, Structured Development for Real-time Systems, Yourdon Press, 1985.

[481] R. Waszkowski, *Low-code platform for automating business processes in manufacturing*, IFAC-PapersOnLine, 52 (2019), pp. 376–381. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019. 10.1016/j.ifacol.2019.10.060.

[482] H. Weigand, *Two decades of language/action perspective*, Natural Language Engineering, 49 (2006), pp. 45–46.

[483] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, *Model-Based System Architecture*, Wiley Series in Systems Engineering and Management, Wiley, 2022. https://books.google.nl/books?id=1GNoEAAAQBAJ.

[484] S. A. White, *Process Modeling Notations and Workflow Patterns*, BP-Trends, (2004).

[485] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*, Springer Berlin, Heidelberg, Dec. 2014. 10.1007/978-3-662-43839-8.

[486] R. Wiersma and P. Ravesteyn, *A method for defining optimum service granularity*, in 21^{st} Annual International Information Management Association (IIMA), Utrecht, The Netherlands, 2010.

[487] G. M. Wijers, *Modelling support in information systems development*, PhD thesis, Delft University of Technology, 1991.

[488] G. M. Wijers and H. Heijes, *Automated support of the modelling process: A view based on experiments with expert information engineers*, in CAiSE 1990: Advanced Information Systems Engineering, B. Steinholtz, A. Sølvberg, and L. Bergman, eds., vol. 436 of Lecture Notes in Computer Science, Springer, Berlin, 1990, pp. 88–108. 10.1007/BFb0000588.

[489] Y. Windarto, R. Hersant, and E. Putro, *Developing Home Service System; Business Process Reengineering for Motorcycle Workshop*, Indonesian Journal of Information Systems, 3 (2021), pp. 94–104. 10.24002/ijis.v3i2.4144.

[490] R. Winter, *Design Solution Analysis for the Construction of Situational Design Methods*, in Engineering Methods in the Service-Oriented Context, J. Ralyté, I. Mirbel, and R. Deneckère, eds., Berlin, Heidelberg, 2011, Springer Berlin Heidelberg, pp. 19–33.

[491] J. J. V. R. Wintraecken, *Informatie-analyse volgens NIAM: in theorie en praktijk*, Academic Service, Den Haag, second ed., 1985. In Dutch.

[492] E. Wolff, *Microservices: Flexible Software Architecture*, Addison-Wesley Professional, 2016.

[493] P. Y. H. Wong and J. Gibbons, *A Process Semantics for BPMN*, in Formal Methods and Software Engineering, S. Liu, T. Maibaum, and K. Araki, eds., vol. 5256 of Lecture Notes in Computer Science, Berlin, Heidelberg, 2008, Springer Berlin Heidelberg, pp. 355–374. 10.1007/978-3-540-88194-0_22.

[494] F. Wu, L. Priscilla, M. Gao, F. Caron, W. De Roover, and J. Vanthienen, *Modeling Decision Structures and Dependencies*, in On the Move to Meaningful Internet Systems: OTM 2012 Workshops., vol. 7567 of Lecture Notes in Computer Science, 2012. 10.1007/978-3-642-33618-8_69.

[495] A. W. Wymore, *Model-Based Systems Engineering*, CRC Press, first ed., 1993. 10.1201/9780203746936.

[496] L. Xu, H. Liu, S. Wang, and K. Wang, *Modelling and analysis techniques for cross-organizational workflow systems*, Systems Research and Behavioral Science, 26 (2009), pp. 367–389. 10.1002/sres.978.

[497] R. K. Yin, *Case study research, Design and Methods.*, vol. 5 of Applied Social Research Methods, Sage Publications, fourth ed., 2009.

[498] E. Yourdon, *Modern Structured Analysis*, Prentice Hall, 1988.

[499] ——, *Just Enough Structured Analysis*, Yourdon, 2006.

[500] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice Hall, 1978.

[501] J. A. Zachman, *Conceptual, logical, physical: It is simple*, Zachman International, 36 (2000).

[502] I. Zayour, I. Moukadem, and I. Moghrabi, *Complexity is in the Brain of the Beholder: A Psychological Perspective on Software Engineering's Ultimate Challenge*, Journal of Software, 8 (2013).

[503] B. P. Zeigler, S. Mittal, and M. K. Traore, *MBSE with/out Simulation: State of the Art and Way Forward*, Systems, 6 (2018). 10.3390/systems6040040.

[504] M. Zgorzelski and P. Zeno, *Flowcharts, data flows, SADT, IDEF and NIAM for enterprise engineering*, in Modelling Techniques for Business Process Re-engineering and Benchmarking, G. Doumeingts and J. Browne, eds., IFIP Advances in Information and Communication Technology, Springer, Boston, MA, 1997. 10.1007/978-0-387-35067-7_7.

[505] J. Zhang and B. H. C. Cheng, *Model-Based Development of Dynamically Adaptive Software*, in Proceedings of the 28th International Conference on Software Engineering, ICSE '06, New York, NY, USA, 2006, Association for Computing Machinery, pp. 371–380. 10.1145/1134285.1134337.

[506] K. X. Zhu and Z. Z. Zhou, *Research Note - Lock-In Strategy in Software Competition: Open-Source Software vs. Proprietary Software*, Information Systems Research, 23 (2011), pp. 536–545. 10.1287/isre.1110.0358.

[507] I. Zikra, J. Stirna, and J. Zdravkovic, *Bringing Enterprise Modeling Closer to Model-Driven Development*, in The Practice of Enterprise Modeling - 4th IFIP WG 8.1 Working Conference (PoEM 2011), P. Johannesson, J. Krogstie, and A. L. Opdahl, eds., vol. 92 of Lecture Notes in Business Information Processing, Springer, Nov. 2011, pp. 268–282. 10.1007/978-3-642-24849-8_20.

[508] O. Zimmermann, *Microservices tenets - Agile approach to service development and deployment*. Online, Springer-Verlag Berlin Heidelberg, Nov. 2016. 10.1007/s00450-016-0337-0.

[509] ——, *Domain Specific Service Decomposition with Microservice API Patterns*. Online, Feb. 2019. Accessed 2021-Nov-11. https://www.conf-micro.services/2019/slides//keynotes/Zimmerman.pdf.

[510] M. Zoet, J. Versendaal, P. Ravesteyn, and R. Welke, *Alignment of Business Process Management and Business Rules*, in ECIS 2011 Proceedings, 2011.

[511] A. Z. Zrnec, M. Bajec, and M. Krisper, *Enterprise modelling with UML*, Electrotechnical Review, 68 (2001), pp. 109–114.

[512] M. zur Muehlen, M. Indulska, and K. Kittel, *Towards Integrated Modeling of Business Processes and Business Rules*, in ACIS 2008 Proceedings, 2008.

List of Figures

List of Tables

List of Acronyms

1GL First-Generation Programming Language 37

2GL Second-Generation Programming Language 37

3GL Third-Generation Programming Language 37, 38

4EM For Enterprise Modeling 53, 57, 220

4GL Fourth-Generation Programming Language 37, 38, 150

5GL Fifth-Generation Programming Language 38, 150

ADM Architecture Development Method 53

ADR Action Design Research xi, xiv, xviii, 13–15, 22–25, 139, 156, 157

AI Artificial Intelligence xv, 6, 155, 159

AM Action Model 62, 65, 67, 86, 88, 115, 128, 150

API Application Programming Interface 41–43, 46–48, 84–87, 89, 92, 94–96, 126, 127, 129, 134, 157

ARIS Architecture of Integrated Information Systems 52–57, 219

ARS Action Rule Specification 65, 86, 88, 91, 92, 104, 115, 116, 119, 120, 127, 209, 210, 229, 231–233, 269

BCI-3D Three Dimensional method for Business Components Identification 76

BPEL Business Process Execution Language 54

BPMN Business Process Modeling and Notation 54, 55, 57, 156, 159, 219, 222

C-act coordination act 59, 60, 62, 63, 65, 69, 76, 86, 88, 94, 95, 100, 103–106, 110, 122, 130, 154

C-fact coordination fact 59, 60, 62, 63, 65, 77

CASE Computer-aided Software Engineering 9, 10, 37

CI/CD Continuous Integration and Continuous Development 8

CM Cooperation Model 60, 65–67, 86, 89, 103–108, 115, 116, 128

CMMN Case Management Modeling and Notation 54

CMS Content Management System 98, 105, 273

CogNIAM Cognition enhanced Natural language Information Analysis Method 50, 54, 57, 219

CPU Central Processing Unit 38

CRUD Create, Read, Update, and Delete 127, 129, 130

CSD Coordination Structure Diagram 65, 89, 90, 100, 118

CTP Complete Transaction Pattern 58–60, 65, 66, 76, 78, 79, 86, 103, 109, 115, 127, 129, 133, 145

DEMO Design and Engineering Methodology for Organizations vii, ix–xi, xiv–xvi, xviii–xx, xxx, xxxi, 54, 55, 57–59, 62, 66, 67, 71, 72, 76, 78, 79, 83, 85–87, 92, 94–100, 103, 105–107, 109–111, 113–117, 122–124, 126–130, 133, 135, 141–150, 152, 153, 155–159, 212, 219, 235, 273

DEMO-SL DEMO Specification Language 55, 66, 86, 158

DFD Data Flow Diagram 9, 49

DFS Derived Fact Specification 65, 86, 88, 91, 101, 119, 127, 129

DMN Decision Modeling and Notation 54, 219, 222

DMS Document Management System 98, 105

DSL Domain Specific Language 38, 40

ECS exploratory case study 15, 24, 26, 141, 157

EIF Enterprise Implementation Framework xi, xiv, 58, 68, 71, 72, 96, 109, 123, 124, 141, 144, 147, 148, 154, 156

EMDSD Enterprise Model-driven Software Development xi, 40, 48, 50, 52, 57–59, 71, 77, 140, 141, 146–148, 154, 155, 159, 273

EPC Event-driven Process Chain 54–57, 219, 223

ER Entity-Relationship 54, 212

ERD Entity-Relationship Diagram 8, 45, 49

ERP Enterprise Resource Planning 98

FM Fact Model 65–67, 86, 104, 105, 115, 119, 127–129

GERAM Generalized Enterprise Reference Architecture and Methodology 52

GSDP General System Development Process 29, 33–35, 37, 40, 48, 58, 68, 71, 79, 98, 99, 140, 144–146, 154, 155, 207

HTTP Hypertext Transfer Protocol 42, 43, 94, 95

IAF Integrated Architecture Framework 52

ICT Information and Communication Technology 3, 84, 213

ICTU Stichting ICT Uitvoeringsorganisatie ix, 26, 83–85, 89

IS Information System 3, 36

ISDM Information Systems Development Method 8

IT Information Technology xvii, xviii, xx, 3, 5, 22, 23, 26, 34, 47, 53, 64, 68, 74, 75, 77, 78, 84, 95, 98, 106, 110, 142, 149, 151, 152, 154, 158, 273

JSON JavaScript Object Notation 66, 79, 92, 96, 130, 145, 158, 235

LLM Large Language Model xv, 155, 159

MBE Model-based Engineering xiii, 2, 9, 10, 29, 35–37, 40, 48, 139, 140, 146, 147

MDA Model-driven Architecture xiii, 9, 35, 143, 156

MDD Model-driven Development 35, 37, 38, 46, 142

MDE Model-driven Engineering 35

MDSD Model-driven Software Development vii, xi, xiv, 35, 37–41, 44, 47–49, 51, 71, 79, 125, 126, 133, 134, 140–146, 153–156, 273

MEMO Multi-Perspective Enterprise Modeling 56, 57, 224

MoD Ministry of Defense ix, 114, 123

MTK Multiple Transaction Kind 127

NIAM Natural language Information Analysis Method 30, 54

NS Normalized System xv, xix, xxx, 7, 15, 26, 29, 43, 44, 47, 48, 71, 76, 78, 94, 113–117, 120–124, 134, 142, 146, 154, 155, 160, 273

O-AA Open Agile Architecture 273

OAS OpenAPI Specification 42, 43, 92–96, 157, 158

OER Organizational Essence Revealing 55, 79, 145

OFD Object Fact Diagram 65, 86, 88, 90, 91, 95, 96, 100, 119

OIV Organization Implementation Variable xiv, xv, xix, 58, 68–72, 79, 109, 123, 126–128, 130, 133–135, 141, 143, 145–148, 157, 160

OMG Object Management Group 54, 56

OMI Open Model Initiative 52

OS Object System 33, 34, 68

P-act production act 59, 65, 76, 86, 100

P-fact production fact 59, 60, 63, 77

PHP PHP: Hypertext Preprocessor 214, 273

PM Process Model 65–67, 103–105, 116, 120, 128

PRINCE2 Projects in Controlled Environments, version 2 52, 156

PSD Process Structure Diagram 65

ROME Return On Modeling Effort 55, 97, 98, 109, 110

SADT Structured Analysis and Design Technique 8, 37, 49

SAFe Scaled Agile Framework 273

SBVR Semantics of Business Vocabulary and Business Rules 54, 56, 57, 219, 225

SDK Software Development Kit 126, 133

SOA Service Oriented Architecture 42, 76, 85

SQL Structured Query Language 273

SSD Structured System Design 8, 37

STD State Transition Diagram 9, 117

SysML System Modeling Language 56, 57

TOGAF The Open Group Architecture Framework 52, 53, 156, 273

TPD Transaction Pattern Diagram 65, 100

TPT Transactor Product Table 65, 89, 90, 101, 118

UI user interface 39–41, 134, 141, 142, 155

UML Unified Modeling Language 9, 35, 50, 52, 54, 56, 57, 143, 156, 159, 225–228

US Using System 33, 34

VSM Viable System Model 33, 144

WIS Work Instruction Specification 65

WoC Way of Controlling 8, 9, 52, 146, 156, 159

WoM Way of Modeling 8, 9, 51, 141, 146

WoS Way of Supporting 8, 9, 51, 141, 146

WoT Way of Thinking 8, 9, 29, 40, 51, 53, 54, 56, 57, 141, 146, 154, 156

WoW Way of Working 8, 9, 51, 53, 55–57, 141, 146

WS-BPEL Web Services Business Process Execution Language 76

XML Extensible Markup Language 53, 66, 79, 145

YAML YAML Ain't Markup Language 42, 92, 237

Part IV

Appendices

A

Enterprise Modeling Techniques - Examples

This chapter contains examples of the discussed enterprise modeling techniques in Chapter 4. Examples for ARIS and CogNIAM are left out; instead, examples for BPMN, DMN, EPC and SBVR are included. Examples for DEMO are left out as they can be found in Section 4.2 and Part II. The given examples are considered self-explanatory. For further information the reader is referred to Section 4.1 and the provided references.

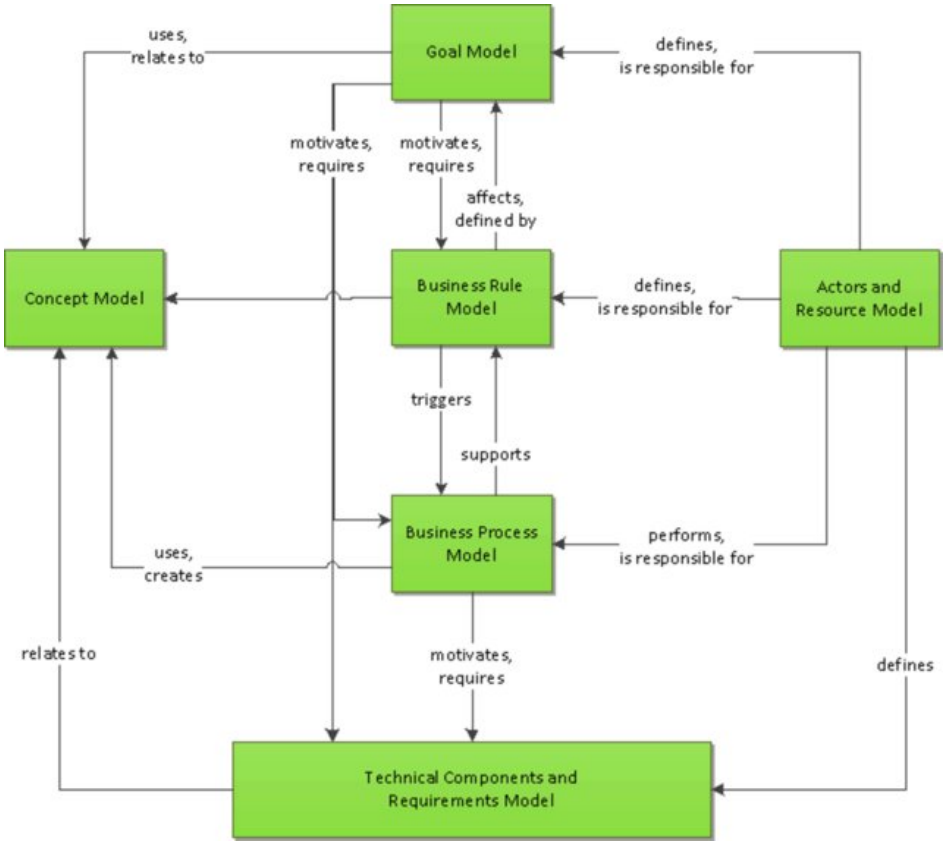


Figure A.1: Sub models of the 4EM approach, taken from [394]

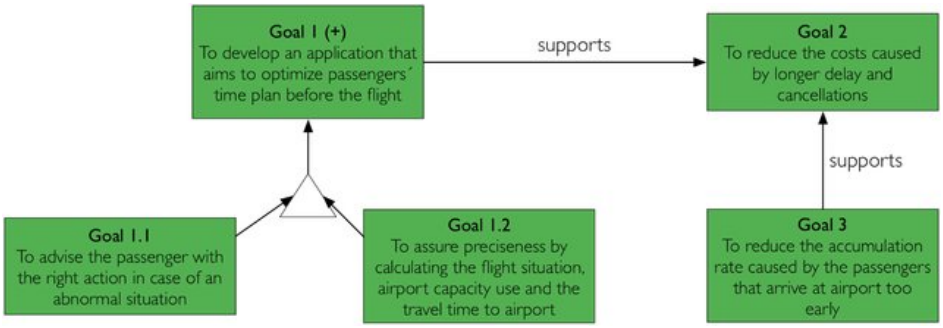


Figure A.2: Example goal model of the 4EM approach, taken from [394]

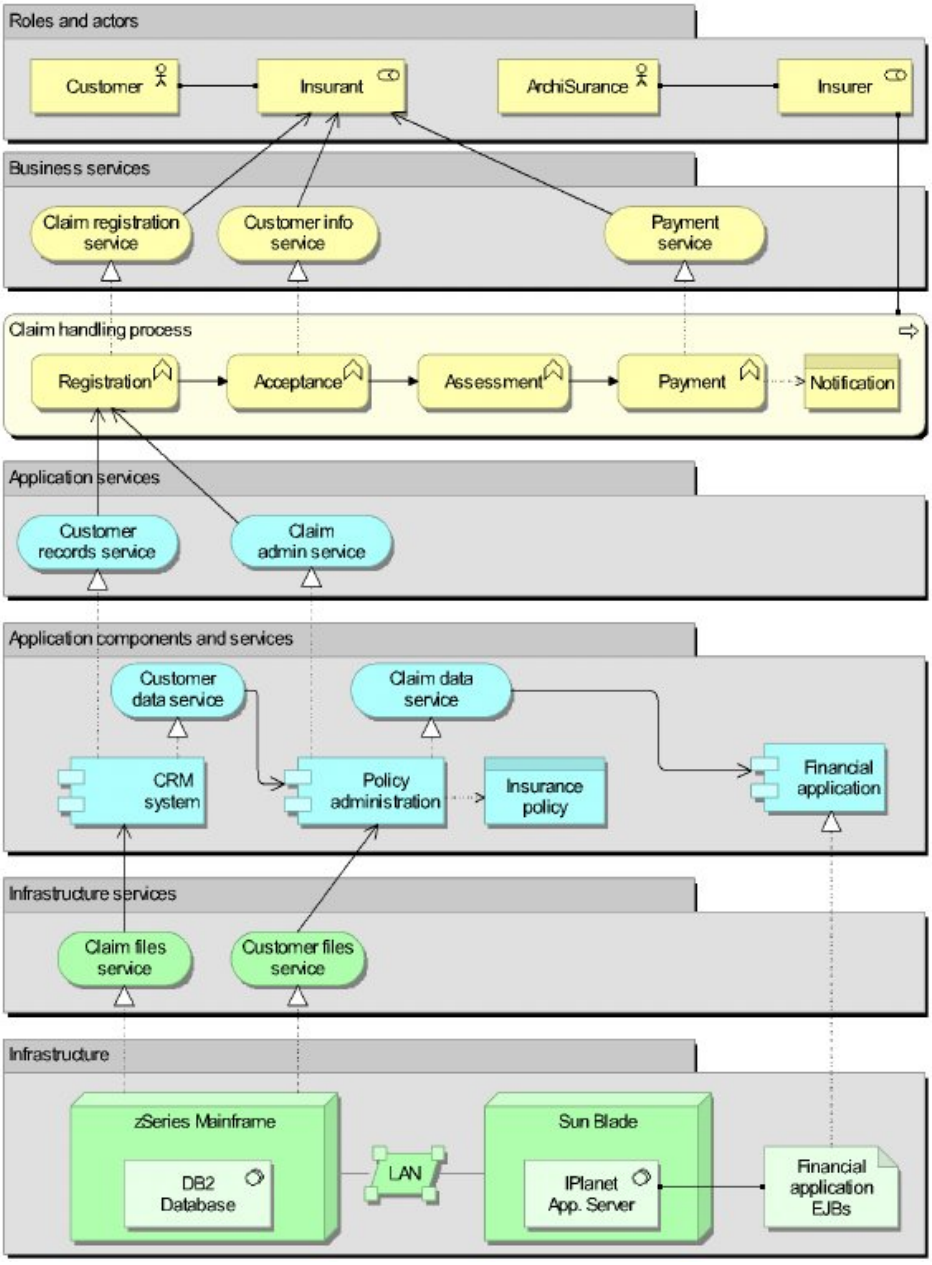


Figure A.3: Example ArchiMate model, taken from [278]

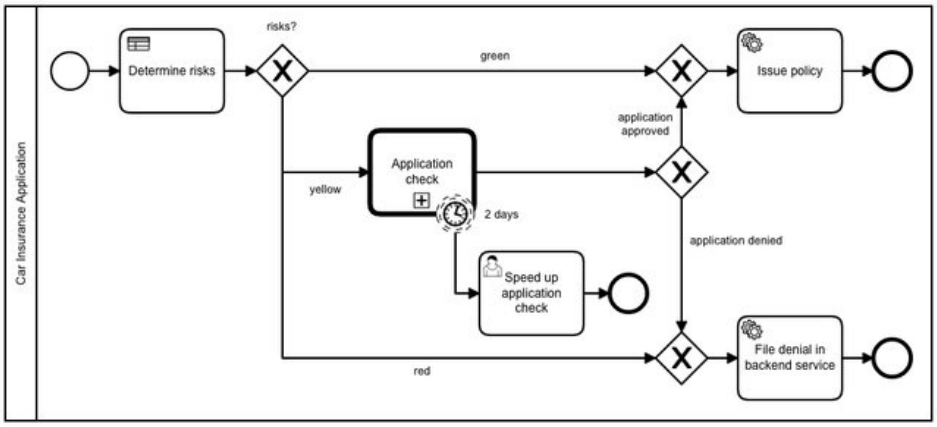
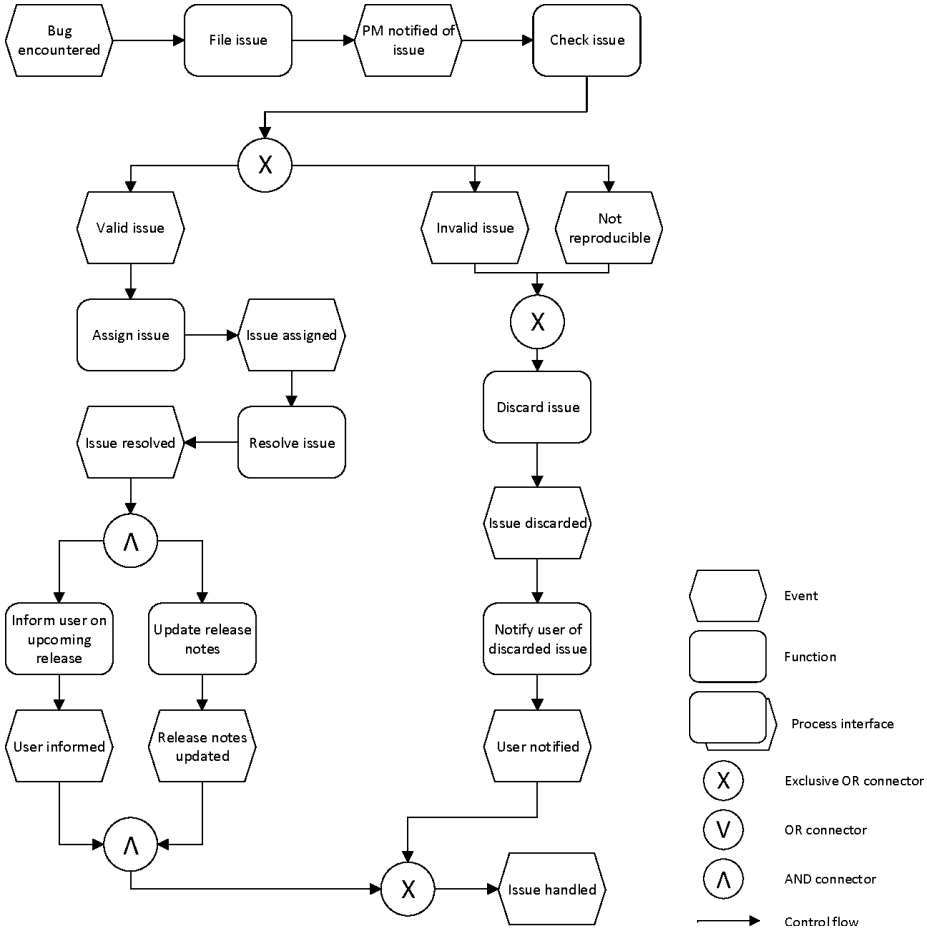


Figure A.4: Example BPMN process model, taken from [108]

C	Input			Output	
	Age	Car Manufacturer	Type of Car	Risk	Risk Evaluation
1	<= 21	–	–	"beginner"	"yellow"
2	<= 30	"BMW"	–	"young and fast"	"yellow"
3	–	"Porsche"	"911"	"careless speeding"	"yellow"
4	–	"BMW"	"X3"	"premium car"	"yellow"
5	<= 25	"Porsche"	"911"	"young and too fast"	"red"

Figure A.5: Example DMN decision table, taken from [108]



(a) Example EPC (b) Legend of EPCs

Figure A.6: Example and legend of EPC, taken from [401]

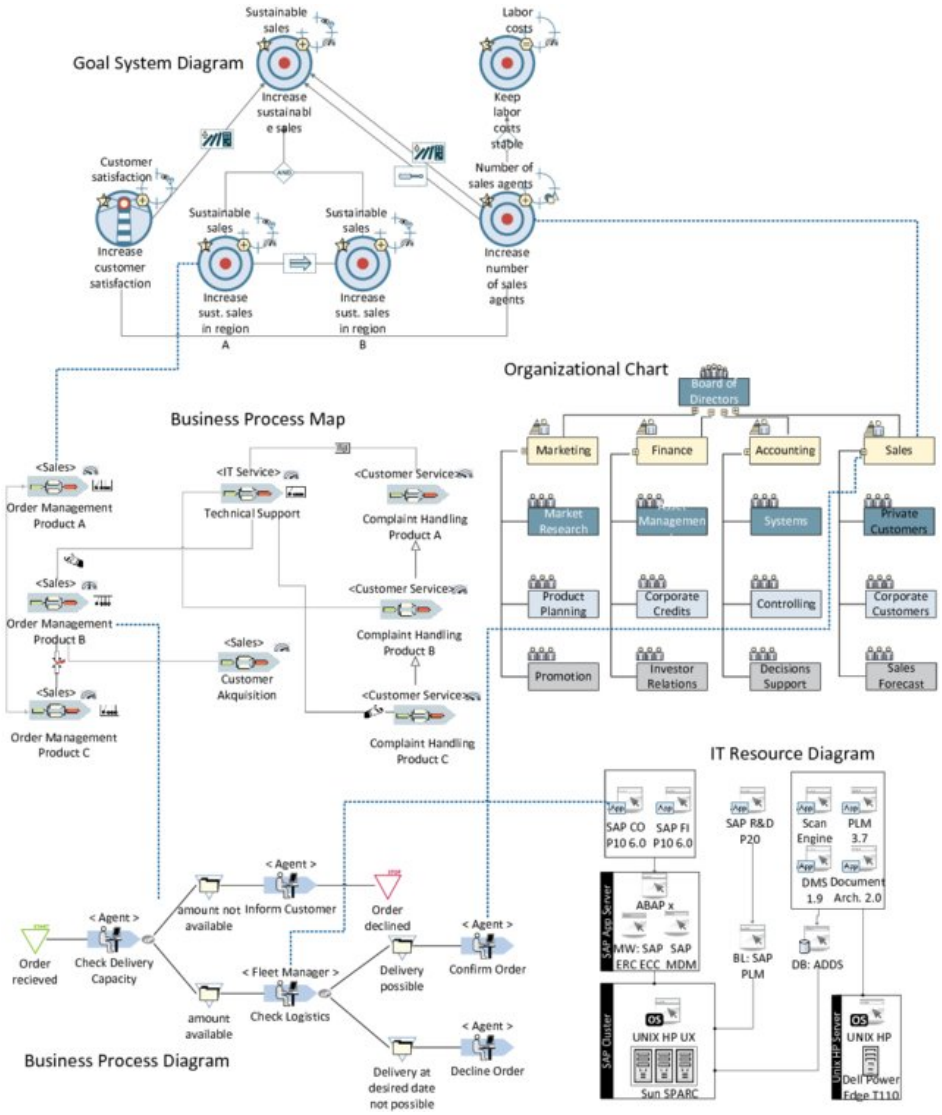


Figure A.7: MEMO example of interrelated diagrams representing an enterprise model, taken from [49]

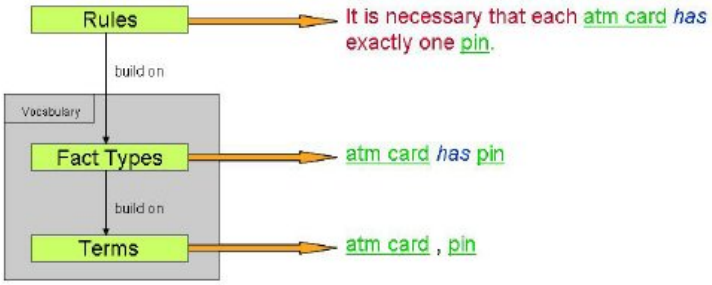


Figure A.8: SBVR schema and example, taken from [366]

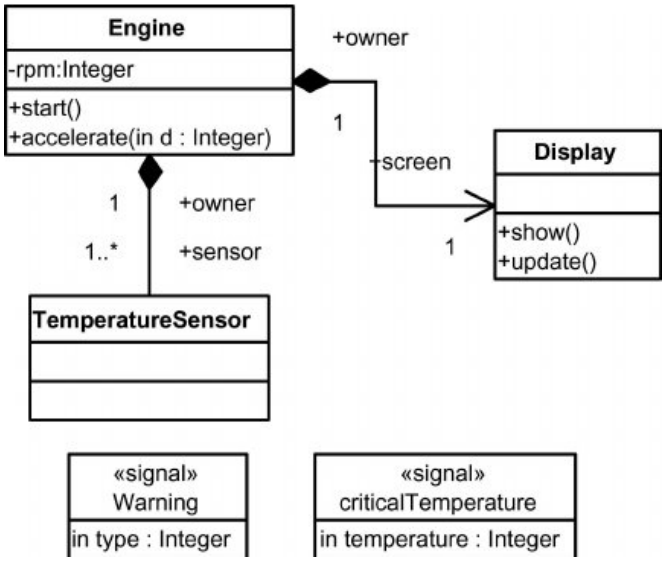


Figure A.9: Example UML class diagram, taken from [180]

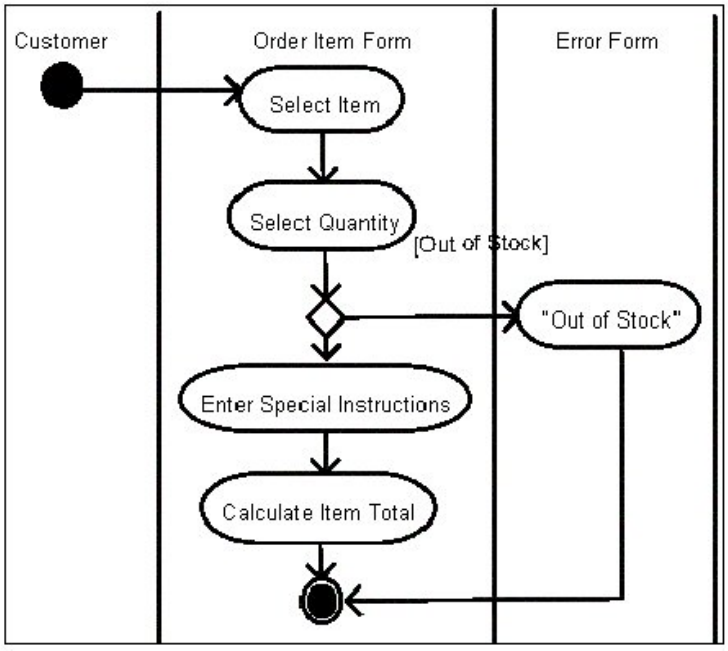


Figure A.10: Example UML activity diagram, taken from [229]

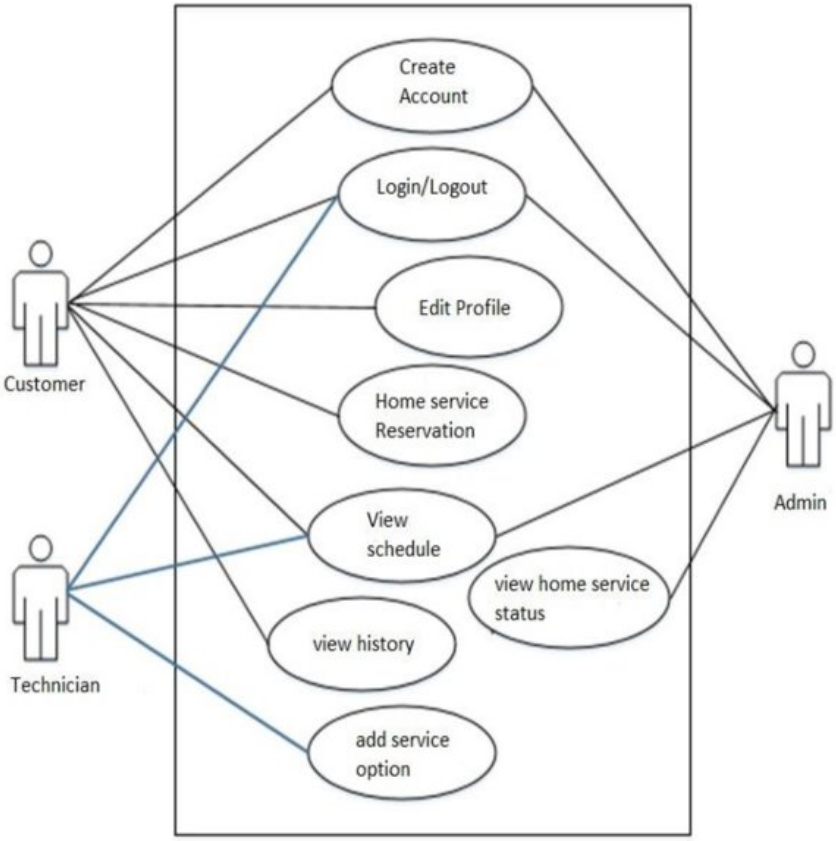


Figure A.11: Example UML use case diagram, taken from [489]

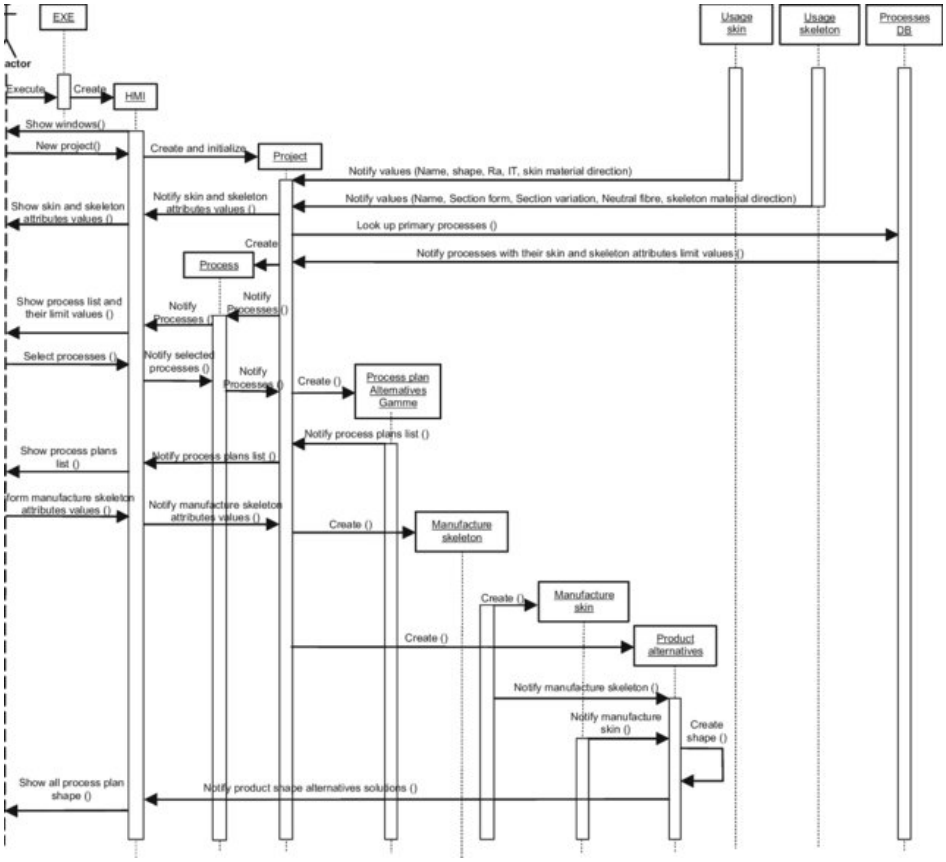


Figure A.12: Example UML sequence diagram, taken from [420]

B

Action Rule Specifications for Social Housing

This addendum lists all (relevant) Action Rule Specifications for the area of registration within the Social Housing Domain.

when registration starting **for** [registration] **is** requested			(TK01/rq)
	with	**the** starting day **of** [registration] **is some** day; **the** member of [registration] **is some** person; **the** payer **of** [registration] **is some** person	
if	*rightness:*	**the** performer **of the** request **is the** member **of** [registration]; **the** addressee **of the** request **is a** registration starter	
	sincerity:	* *no specific condition* *	
	truth:	**the** age **of** member **of** [registration] **on** starting day **of** [registration] **is greater than or equal to** 18; nationality **of** member **of** [registration] **is** Dutch; **NOT** member **of** [registration] has active registration **on the** starting day **of** [registration]; **the** year **of the** starting day **of** [registration] **is greater than or equal to the** year **of** Now	
if	*performing the action after* ***then*** *is considered justifiable*		
then	promise	registration starting **for** [registration] **to the** performer **of the** request	[TK01/pm]
else	decline	registration starting **for** [registration] **to the** performer **of the** request **with** * *reason for declining* *	[TK01/dc]

Table B.1: ARS01 (executed by AR01)

when registration starting **for** [registration] **is** promised			(TK01/pm)
if	*rightness:*	**the** performer **of the** promise **is a** registration starter; **the** addressee **of the** promise **is the** member **of** [registration]	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	request	registration paying **for** [registration] **in the** year **of the** starting day **of** [registration] **to the** payer **of** [registration] **with the** requested paid amount **of** [registration] **in the** year **of the** starting day **of** [registration] **is equal to the** standard registration fee **in the** year **of the** starting day **of** [registration]	[TK02/rq]

Table B.2: ARS02 (executed by AR01)

when registration paying **for** [registration] **in** [year] **is** declared			(TK02/da)
if	*rightness:*	**the** performer **of the** declaration **is the** payer **of** [registration]; **the** addressee **of the** declaration **is the** registration starter **of** [registration]	
	sincerity:	* *no specific condition* *	
	truth:	**the** declared paid amount **of** registration paying **for** [registration] **in** [year] **is equal to the** requested paid amount **of** registration paying **for** [registration] **in** [year]	
if	*performing the action after* ***then*** *is considered justifiable*		
then	accept	registration paying **for** [registration] in [year] **to the** performer **of the** declaration	[TK02/ac]
else	reject	registration paying **for** [registration] in [year] **to the** performer **of the** declaration **with** * *reason for rejecting* *	[TK02/rj]

Table B.3: ARS03 (executed by AR02)

when registration starting **for** [registration] **is** promised			(TK01/pm)
	while	registration paying **for** [registration] **in the** year **of the** starting day **of** [registration] **is** accepted	(TK02/ac)
if	*rightness:*	* *no specific condition* *	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	execute	registration starting **for** [registration];	[TK01/ex]
	declare	registration starting **for** [registration] **to the** addressee **of the** promise	[TK01/da]

Table B.4: ARS04 (executed by AR01)

when registration ending **for** [registration] **is** requested			(TK03/rq)
	with	**the** ending day **of** [registration] **is some** day	
if	*rightness:*	**the** performer **of the** request **is the** member **of** [registration] **or the** performer **of the** request **is a** registration manager; **the** addressee **of the** request **is a** registration ender	
	sincerity:	* *no specific condition* *	
	truth:	**the** ending day **of** [registration] **is greater than or equal to the** starting day **of** [registration]	
if	*performing the action after* ***then*** *is considered justifiable*		
then	promise	registration ending **for** [registration] **to the** performer **of the** request	[TK03/pm]
then	decline	registration ending **for** [registration] **to the** performer **of the** request **with** * *reason for declining* *	[TK03/dc]

Table B.5: ARS05 (executed by AR03)

when registration ending **for** [registration] **is** promised			(TK03/pm)
if	*rightness:*	**the** performer **of the** promise **is a** registration ender; **the** addressee **of the** promise **is the** member **of** [registration] **or the** addressee **of the** promise **is a** registration manager	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	execute	registration ending **for** [registration];	[TK03/ex]
	declare	registration ending **for** [registration] **to the** addressee **of the** promise	[TK03/da]

Table B.6: ARS06 (executed by AR03)

when registration management **for** [year] **is** requested			(TK04/rq)
if	*rightness:*	**the** performer **of the** request **is the** registration manager **of** [year] **minus** 1; **the** addressee **of the** request **is the** registration manager **of** [year]	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	promise	registration management **for** [year] **to the** performer **of the** request;	[TK04/pm]
	request	registration management **for** [year] **plus** 1 **to the** registration manager **of** [year] **plus** 1	[TK04/rq]
else	decline	registration management **for** [year] **to the** performer **of the** request **with** * *reason for declining* *;	[TK04/dc]
	request	registration management **for** [year] **plus** 1 **to the** registration manager **of** [year] **plus** 1	[TK04/rq]

Table B.7: ARS07 (executed by AR04)

when registration management **for** [year] **is** promised			(TK04/pm)
if	*rightness:*	**the** performer **of the** promise **is the** registration manager **of** [year]; **the** addressee **of the** promise **is the** registration manager **of** [year]	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	**for each** [registration] **choose:**		
	to-be-renewed:		
	request	registration paying **for** [registration] **in** [year] **to the** payer **of** [registration] **with the** requested paid amount **of** [registration] **in** [year] **is equal to the** standard renewal fee **in** [year]	(TK02/rq)
	to-be-ended:		
	request	registration ending **for** [registration] **to some** registration ender **with the** ending day **of** [registration] **is** Now	(TK03/rq)
	other:	* *no action needed* *	

Table B.8: ARS08 (executed by AR04)

when registration ending **for** [registration] **is** declared			(TK03/da)
if	*rightness:*	**the** performer **of the** declaration **is the** payer **of** [registration]; **the** addressee **of the** declaration **is the** registration starter **of** [registration] **in** [year]	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	accept	registration paying **for** [registration] in [year] **to the** performer **of the** declaration	[TK02/ac]
else	reject	registration paying **for** [registration] in [year] **to the** performer **of the** declaration **with** * *reason for rejecting* *	[TK02/rj]

Table B.9: ARS09 (executed by AR04)

when registration management **for** [year] **is** promised			(TK04/pm)
	while for each [registration] **in** registrations **in** [year] **choose:**		
	to-be-renewed:	registration paying **for** [registration] **in** [year] **is** accepted	(TK02/ac)
	to-be-ended:	registration ending **for** [registration] **is** accepted	(TK03/ac)
	other:	* *no specific condition* *	
if	*rightness:*	**the** performer **of the** promise **is the** registration manager **of** [year]; **the** addressee **of the** promise **is the** registration manager **of** [year]	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	execute	registration management **for** [year];	(TK04/ex)
	declare	registration management **for** [year] **to the** addressee **of the** promise	(TK04/da)

Table B.10: ARS10 (executed by AR04)

when registration management **for** [year] **is** declared			(TK04/da)
if	*rightness:*	**the** performer **of the** declaration **is the** registration manager **of** [year]; **the** addressee **of the** declaration **is the** registration manager **of** [year]	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	accept	registration management **for** [year] **to the** performer **of the** declaration	[TK04/ac]
else	reject	registration management **for** [year] **to the** performer **of the** declaration **with** * *reason for rejecting* *	[TK04/rj]

Table B.11: ARS11 (executed by AR04)

C

JSON file for Social Housing

This addendum shows the (manually) created JSON file representing the DEMO model of the area of registration within the Social Housing Domain.


```
{
    "transactionkinds": [
        {"id": "TK01", "name": "registration starting", "type": "elementary", "in": true, "casekinds": ["
            Registration"], "product": "registration is started", "productname": "StartedRegistration"},
        {"id": "TK02", "name": "registration paying", "type": "elementary", "in": false, "casekinds": ["
            Registration", "YearE"], "product": "the fee for registration in year is paid", "productname": "
            AnnualRegistrationPayment"},
        {"id": "TK03", "name": "registration ending", "type": "elementary", "in": true, "casekinds": ["
            Registration"], "product": "registration is ended", "productname": "EndedRegistration"},
        {"id": "TK04", "name": "registration management", "type": "elementary", "in": true, "casekinds": ["
            YearE"], "product": "registration management for year is done", "productname": "TK04product"}
    ],
    "actorroles": [
        {"id": "CTAR01", "name": "(aspirant) member", "type": "composite"},
        {"id": "AR01", "name": "registration starter", "type": "elementary", "focus": "in"},
        {"id": "AR02", "name": "registration payer", "type": "elementary", "focus": "out"},
        {"id": "AR03", "name": "registration ender", "type": "elementary", "focus": "in"},
        {"id": "AR04", "name": "registration manager", "type": "elementary", "focus": "in"}

    ],
    "factkinds": [
        {"name": "Registration", "type": "entitytype", "focus": "in"},
        {"name": "Person", "type": "entitytype", "focus": "out"},
        {"name": "YearV", "type": "valuetype", "primitive": "integer"},
        {"name": "YearE", "type": "entitytype", "focus": "out"},
        {"name": "Nationality", "type": "valuetype", "values": "NL, EN", "primitive": "string"},
        {"name": "Day", "type": "valuetype", "primitive": "datetime"},
        {"name": "Money", "type": "valuetype", "primitive": "number"},
        {"name": "member", "type": "propertytype", "domain": "Registration", "range": "Person"},
        {"name": "payer", "type": "propertytype", "domain": "Registration", "range": "Person"},
        {"name": "starting day", "type": "attributetype", "domain": "StartedRegistration", "range": "Day"},
        {"name": "ending day", "type": "attributetype", "domain": "EndedRegistration", "range": "Day"},
        {"name": "year", "type": "attributetype", "domain": "YearE", "range": "YearV"},
        {"name": "standard registration fee", "type": "attributetype", "domain": "YearE", "range": "Money"},
        {"name": "standard renewal fee", "type": "attributetype", "domain": "YearE", "range": "Money"},
        {"name": "day of birth", "type": "attributetype", "domain": "Person", "range": "Day"},
        {"name": "paid amount", "type": "attributetype", "domain": "AnnualRegistrationPayment", "range": "
            Money"},
        {"name": "nationality", "type": "attributetype", "domain": "Person", "range": "Nationality"},
        {"name": "PersonAge", "type": "derived", "parameters": ["Person", "Day"], "result": "primitive:integer
            "},
        {"name": "PersonHasActiveRegistration", "type": "derived", "parameters": ["Person", "Day"], "result":
            "primitive:boolean"}
    ],
    "actionrules":
    [
        {"id": "ARS01", "actorrole": "AR01", "when": "TK01rq", "while": [], "respond": ["TK01pm"], "
            respondelse": ["TK01dc"]},
```

```
        {"id": "ARS02", "actorrole": "AR01", "when": "TK01pm", "while": [], "respond": ["TK02rq"]},
        {"id": "ARS03", "actorrole": "AR01", "when": "TK02da", "while": [], "respond": ["TK02ac"], "
            respondelse": ["TK02rj"]},
        {"id": "ARS04", "actorrole": "AR01", "when": "TK01pm", "while": ["TK02ac"], "respond": ["TK01ex", "TK0
            1da"]},
        {"id": "ARS05", "actorrole": "AR03", "when": "TK03rq", "while": [], "respond": ["TK03pm"], "
            respondelse": ["TK03dc"]},
        {"id": "ARS06", "actorrole": "AR03", "when": "TK03pm", "while": [], "respond": ["TK03ex", "TK03da"]},
        {"id": "ARS07", "actorrole": "AR04", "when": "TK04rq", "while": [], "respond": ["TK04pm", "TK04rq"], "
            respondelse": ["TK04dc", "TK04rq"]},
        {"id": "ARS08", "actorrole": "AR04", "when": "TK04pm", "while": [], "respond" : ["TK02rq, TK03rq"]},
        {"id": "ARS09", "actorrole": "AR04", "when": "TK03da", "while": [], "respond": ["TK03ac"], "
            respondelse": ["TK03rj"]},
        {"id": "ARS10", "actorrole": "AR04", "when": "TK04pm", "whileall": ["TK02ac", "TK03ac"], "respond": ["
            TK04ex", "TK04da"]},
        {"id": "ARS11", "actorrole": "AR04", "when": "TK04da", "while": [], "respond": ["TK04ac"], "
            respondelse": ["TK04rj"]}
    ],
    "oivs": [
        {"name": "functionary type"},
        {"name": "organizational unit"},
        {"name": "authorization"}
    ]
}
```

D

Generated YAML file for Social Housing

This addendum lists the generated YAML Ain't Markup Language (YAML) file for the area of registration within the Social Housing Domain.

```
openapi: 3.0.0
info:
 version: 1.0.0
 title: test
paths:
 /TK01rq:
  post:
   summary: creates a new TK01-rq
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/StartedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
 /TK01rv-rq:
  post:
   summary: creates a new TK01-rv-rq
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/StartedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
```

```
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK01rv-rq-al:
 post:
  summary: creates a new TK01-rv-rq-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01rv-rq-rf:
 post:
  summary: creates a new TK01-rv-rq-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01pm:
 post:
  summary: creates a new TK01-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01rv-pm:
 post:
  summary: creates a new TK01-rv-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
```

```
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK01rv-pm-al:
 post:
  summary: creates a new TK01-rv-pm-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01rv-pm-rf:
 post:
  summary: creates a new TK01-rv-pm-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01da:
 post:
  summary: creates a new TK01-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01rv-da:
 post:
  summary: creates a new TK01-rv-da
  requestBody:
   required: true
   content:
    application/json:
```

```
      schema:
        $ref: '#/components/schemas/StartedRegistration'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK01rv-da-al:
   post:
    summary: creates a new TK01-rv-da-al
    requestBody:
     required: true
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/StartedRegistration'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK01rv-da-rf:
   post:
    summary: creates a new TK01-rv-da-rf
    requestBody:
     required: true
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/StartedRegistration'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK01ac:
   post:
    summary: creates a new TK01-ac
    requestBody:
     required: true
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/StartedRegistration'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK01rv-ac:
   post:
    summary: creates a new TK01-rv-ac
    requestBody:
```

```
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/StartedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK01rv-ac-al:
 post:
  summary: creates a new TK01-rv-ac-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01rv-ac-rf:
 post:
  summary: creates a new TK01-rv-ac-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01dc:
 post:
  summary: creates a new TK01-dc
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/StartedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK01rj:
```

```
  post:
   summary: creates a new TK01-rj
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/StartedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
 /TK01ex:
  post:
   summary: creates a new TK01-ex
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/StartedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
 /TK02rq:
  post:
   summary: creates a new TK02-rq
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/AnnualRegistrationPayment'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
 /TK02rv-rq:
  post:
   summary: creates a new TK02-rv-rq
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/AnnualRegistrationPayment'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
```

```
    '400':
     description: invalid input
/TK02rv-rq-al:
 post:
  summary: creates a new TK02-rv-rq-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02rv-rq-rf:
 post:
  summary: creates a new TK02-rv-rq-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02pm:
 post:
  summary: creates a new TK02-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02rv-pm:
 post:
  summary: creates a new TK02-rv-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
```

```
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK02rv-pm-al:
 post:
  summary: creates a new TK02-rv-pm-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02rv-pm-rf:
 post:
  summary: creates a new TK02-rv-pm-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02da:
 post:
  summary: creates a new TK02-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02rv-da:
 post:
  summary: creates a new TK02-rv-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
```

```
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK02rv-da-al:
 post:
  summary: creates a new TK02-rv-da-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02rv-da-rf:
 post:
  summary: creates a new TK02-rv-da-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02ac:
 post:
  summary: creates a new TK02-ac
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK02rv-ac:
 post:
  summary: creates a new TK02-rv-ac
  requestBody:
   required: true
   content:
    application/json:
```

```
      schema:
        $ref: '#/components/schemas/AnnualRegistrationPayment'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK02rv-ac-al:
   post:
    summary: creates a new TK02-rv-ac-al
    requestBody:
     required: true
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/AnnualRegistrationPayment'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK02rv-ac-rf:
   post:
    summary: creates a new TK02-rv-ac-rf
    requestBody:
     required: true
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/AnnualRegistrationPayment'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK02dc:
   post:
    summary: creates a new TK02-dc
    requestBody:
     required: true
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/AnnualRegistrationPayment'
    responses:
      '201':
      description: C-fact created
      content:
        application\json:
        schema:
          $ref: '#/components/schemas/CFact'
      '400':
      description: invalid input
  /TK02rj:
   post:
    summary: creates a new TK02-rj
    requestBody:
```

```
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/AnnualRegistrationPayment'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK02ex:
 post:
  summary: creates a new TK02-ex
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rq:
 post:
  summary: creates a new TK03-rq
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-rq:
 post:
  summary: creates a new TK03-rv-rq
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-rq-al:
```

```
  post:
   summary: creates a new TK03-rv-rq-al
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/EndedRegistration'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK03rv-rq-rf:
 post:
  summary: creates a new TK03-rv-rq-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03pm:
 post:
  summary: creates a new TK03-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-pm:
 post:
  summary: creates a new TK03-rv-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
```

```
    '400':
     description: invalid input
/TK03rv-pm-al:
 post:
  summary: creates a new TK03-rv-pm-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-pm-rf:
 post:
  summary: creates a new TK03-rv-pm-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03da:
 post:
  summary: creates a new TK03-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-da:
 post:
  summary: creates a new TK03-rv-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
```

```
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK03rv-da-al:
 post:
  summary: creates a new TK03-rv-da-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-da-rf:
 post:
  summary: creates a new TK03-rv-da-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03ac:
 post:
  summary: creates a new TK03-ac
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-ac:
 post:
  summary: creates a new TK03-rv-ac
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
```

```
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK03rv-ac-al:
 post:
  summary: creates a new TK03-rv-ac-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rv-ac-rf:
 post:
  summary: creates a new TK03-rv-ac-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03dc:
 post:
  summary: creates a new TK03-dc
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK03rj:
 post:
  summary: creates a new TK03-rj
  requestBody:
   required: true
   content:
    application/json:
```

```
      schema:
        $ref: '#/components/schemas/EndedRegistration'
    responses:
     '201':
      description: C-fact created
      content:
       application\json:
        schema:
          $ref: '#/components/schemas/CFact'
     '400':
      description: invalid input
/TK03ex:
 post:
  summary: creates a new TK03-ex
  requestBody:
   required: true
   content:
    application/json:
     schema:
       $ref: '#/components/schemas/EndedRegistration'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
        $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rq:
 post:
  summary: creates a new TK04-rq
  requestBody:
   required: true
   content:
    application/json:
     schema:
       $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
        $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-rq:
 post:
  summary: creates a new TK04-rv-rq
  requestBody:
   required: true
   content:
    application/json:
     schema:
       $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
        $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-rq-al:
 post:
  summary: creates a new TK04-rv-rq-al
  requestBody:
```

```
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/TK04product'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK04rv-rq-rf:
 post:
  summary: creates a new TK04-rv-rq-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04pm:
 post:
  summary: creates a new TK04-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-pm:
 post:
  summary: creates a new TK04-rv-pm
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-pm-al:
```

```
  post:
   summary: creates a new TK04-rv-pm-al
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/TK04product'
   responses:
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK04rv-pm-rf:
 post:
  summary: creates a new TK04-rv-pm-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04da:
 post:
  summary: creates a new TK04-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-da:
 post:
  summary: creates a new TK04-rv-da
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
```

```
    '400':
     description: invalid input
/TK04rv-da-al:
 post:
  summary: creates a new TK04-rv-da-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-da-rf:
 post:
  summary: creates a new TK04-rv-da-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04ac:
 post:
  summary: creates a new TK04-ac
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-ac:
 post:
  summary: creates a new TK04-rv-ac
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
```

```
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK04rv-ac-al:
 post:
  summary: creates a new TK04-rv-ac-al
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rv-ac-rf:
 post:
  summary: creates a new TK04-rv-ac-rf
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04dc:
 post:
  summary: creates a new TK04-dc
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/TK04rj:
 post:
  summary: creates a new TK04-rj
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
```

```
    '201':
     description: C-fact created
     content:
      application\json:
       schema:
        $ref: '#/components/schemas/CFact'
    '400':
     description: invalid input
/TK04ex:
 post:
  summary: creates a new TK04-ex
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/TK04product'
  responses:
   '201':
    description: C-fact created
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/CFact'
   '400':
    description: invalid input
/Registration/{RegistrationID}:
 get:
  summary: returns Registration data for given ID if it exists
  parameters:
   - in: path
     name: RegistrationID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: Registration found
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/Registration'
   '400':
    description: Registration not found
/Registration:
 post:
  summary: creates Registration
  requestBody:
   required: true
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/Registration'
  responses:
   '200':
    description: Registration created
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: invalid input
/Person/{PersonID}:
 get:
  summary: returns Person data for given ID if it exists
  parameters:
   - in: path
     name: PersonID
     schema:
```

```
      type: integer
     required: true
  responses:
   '200':
    description: Person found
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/Person'
   '400':
    description: Person not found
/YearE/{YearEID}:
 get:
  summary: returns YearE data for given ID if it exists
  parameters:
   - in: path
     name: YearEID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: YearE found
    content:
     application\json:
      schema:
       $ref: '#/components/schemas/YearE'
   '400':
    description: YearE not found
/calculatePersonAge:
 get:
  summary: returns PersonAge data for given input
  parameters:
   - in: query
     name: Person
     schema:
      $ref: '#/components/schemas/Person'
     required: true
   - in: query
     name: Day
     schema:
      $ref: '#/components/schemas/Day'
     required: true
  responses:
   '200':
    description: PersonAge calculated
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: invalid input
/calculatePersonHasActiveRegistration:
 get:
  summary: returns PersonHasActiveRegistration data for given input
  parameters:
   - in: query
     name: Person
     schema:
      $ref: '#/components/schemas/Person'
     required: true
   - in: query
     name: Day
     schema:
      $ref: '#/components/schemas/Day'
     required: true
  responses:
   '200':
    description: PersonHasActiveRegistration calculated
```

```
    content:
     application\json:
      schema:
       type: boolean
   '400':
    description: invalid input
/assessARS01/{cactID}:
 get:
  summary: evaluates assess part of ARS01 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS01:
 post:
  summary: performs the response part of ARS01 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS02/{cactID}:
 get:
  summary: evaluates assess part of ARS02 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS02:
 post:
```

```
   summary: performs the response part of ARS02 for given C-act and decision
   requestBody:
    required: true
    content:
     application/json:
      schema:
       type: object
       properties:
        cact:
         $ref: '#/components/schemas/CFact'
        decision:
         type: integer
   responses:
    '201':
     description: response part performed
     content:
      application\json:
       schema:
        type: array
        items:
         type: integer
    '400':
     description: performing response part failed
 /assessARS03/{cactID}:
  get:
   summary: evaluates assess part of ARS03 for given C-act
   parameters:
    - in: path
      name: cactID
      schema:
       type: integer
      required: true
   responses:
    '200':
     description: assessment performed with result
     content:
      application\json:
       schema:
        type: integer
    '400':
     description: assessment failed
 /responseARS03:
  post:
   summary: performs the response part of ARS03 for given C-act and decision
   requestBody:
    required: true
    content:
     application/json:
      schema:
       type: object
       properties:
        cact:
         $ref: '#/components/schemas/CFact'
        decision:
         type: integer
   responses:
    '201':
     description: response part performed
     content:
      application\json:
       schema:
        type: array
        items:
         type: integer
    '400':
     description: performing response part failed
 /assessARS04/{cactID}:
  get:
   summary: evaluates assess part of ARS04 for given C-act
```

```
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS04:
 post:
  summary: performs the response part of ARS04 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS05/{cactID}:
 get:
  summary: evaluates assess part of ARS05 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS05:
 post:
  summary: performs the response part of ARS05 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
```

```
          $ref: '#/components/schemas/CFact'
        decision:
         type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS06/{cactID}:
 get:
  summary: evaluates assess part of ARS06 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS06:
 post:
  summary: performs the response part of ARS06 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS07/{cactID}:
 get:
  summary: evaluates assess part of ARS07 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
```

```
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS07:
 post:
  summary: performs the response part of ARS07 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS08/{cactID}:
 get:
  summary: evaluates assess part of ARS08 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS08:
 post:
  summary: performs the response part of ARS08 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
```

```
        type: array
        items:
         type: integer
    '400':
     description: performing response part failed
/assessARS09/{cactID}:
 get:
  summary: evaluates assess part of ARS09 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS09:
 post:
  summary: performs the response part of ARS09 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS10/{cactID}:
 get:
  summary: evaluates assess part of ARS10 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS10:
 post:
  summary: performs the response part of ARS10 for given C-act and decision
```

```
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/assessARS11/{cactID}:
 get:
  summary: evaluates assess part of ARS11 for given C-act
  parameters:
   - in: path
     name: cactID
     schema:
      type: integer
     required: true
  responses:
   '200':
    description: assessment performed with result
    content:
     application\json:
      schema:
       type: integer
   '400':
    description: assessment failed
/responseARS11:
 post:
  summary: performs the response part of ARS11 for given C-act and decision
  requestBody:
   required: true
   content:
    application/json:
     schema:
      type: object
      properties:
       cact:
        $ref: '#/components/schemas/CFact'
       decision:
        type: integer
  responses:
   '201':
    description: response part performed
    content:
     application\json:
      schema:
       type: array
       items:
        type: integer
   '400':
    description: performing response part failed
/agendaForCTAR01:
 get:
  summary: retrieves agenda for CTAR01
  responses:
```

```
    '200':
     description: agenda for CTAR01 retrieved
     content:
      application\json:
       schema:
        type: array
        items:
         $ref: '#/components/schemas/CFact'
 /agendaForAR01:
  get:
   summary: retrieves agenda for AR01
   responses:
    '200':
     description: agenda for AR01 retrieved
     content:
      application\json:
       schema:
        type: array
        items:
         $ref: '#/components/schemas/CFact'
 /agendaForAR02:
  get:
   summary: retrieves agenda for AR02
   responses:
    '200':
     description: agenda for AR02 retrieved
     content:
      application\json:
       schema:
        type: array
        items:
         $ref: '#/components/schemas/CFact'
 /agendaForAR03:
  get:
   summary: retrieves agenda for AR03
   responses:
    '200':
     description: agenda for AR03 retrieved
     content:
      application\json:
       schema:
        type: array
        items:
         $ref: '#/components/schemas/CFact'
 /agendaForAR04:
  get:
   summary: retrieves agenda for AR04
   responses:
    '200':
     description: agenda for AR04 retrieved
     content:
      application\json:
       schema:
        type: array
        items:
         $ref: '#/components/schemas/CFact'
components:
 schemas:
  CFact:
   type: object
   properties:
    performer:
     type: string
     example: 'Martin'
    addressee:
     type: string
     example: 'Erik'
    intention:
     type: string
```

```
    example: rq
   product:
    $ref: '#/components/schemas/ProductKind'
ProductKind:
 type: object
 oneOf:
  - type: object
    properties:
     StartedRegistration:
      $ref: '#/components/schemas/StartedRegistration'
  - type: object
    properties:
     AnnualRegistrationPayment:
      $ref: '#/components/schemas/AnnualRegistrationPayment'
  - type: object
    properties:
     EndedRegistration:
      $ref: '#/components/schemas/EndedRegistration'
  - type: object
    properties:
     TK04product:
      $ref: '#/components/schemas/TK04product'
StartedRegistration:
 type: object
 properties:
  Registration:
   $ref: '#/components/schemas/Registration'
  starting day:
    $ref: '#/components/schemas/Day'
AnnualRegistrationPayment:
 type: object
 properties:
  Registration:
   $ref: '#/components/schemas/Registration'
  YearE:
   $ref: '#/components/schemas/YearE'
  paid amount:
    $ref: '#/components/schemas/Money'
EndedRegistration:
 type: object
 properties:
  Registration:
   $ref: '#/components/schemas/Registration'
  ending day:
    $ref: '#/components/schemas/Day'
TK04product:
 type: object
 properties:
  YearE:
   $ref: '#/components/schemas/YearE'
Registration:
 type: object
 properties:
  member:
    $ref: '#/components/schemas/Person'
  payer:
    $ref: '#/components/schemas/Person'
Person:
 type: object
 properties:
  day of birth:
    $ref: '#/components/schemas/Day'
  nationality:
    $ref: '#/components/schemas/Nationality'
YearV:
 type: integer
YearE:
 type: object
 properties:
```

```
    year:
      $ref: '#/components/schemas/YearV'
    standard registration fee:
      $ref: '#/components/schemas/Money'
    standard renewal fee:
      $ref: '#/components/schemas/Money'
Nationality:
  type: string
Day:
  type: string
  format: date-time
Money:
  type: number
```

E

Action Rule Specifications for Subsidy Granting

This addendum lists all (relevant) Action Rule Specifications for AR01 (subsidy granter).

when	subsidy granting **for** [subsidy] **is** requested		(TK01/rq)
	with	*check relevant details*	
if	*rightness:*	**the** performer **of the** request **is** *someone related to the subsidy*	
		the addressee **of the** request **is a** subsidy granter	
	sincerity:	* *no specific condition* *	
	truth:	**the** calculated verdict **of** subsidy **is equal to** 'positive'	
if	*performing the action after* ***then*** *is considered justifiable*		
then	request	subsidy evaluation **for** [subsidy]	[TK02/rq]
		to a subsidy evaluator	
else	decline	subsidy granting **for** [subsidy]	[TK01/dc]
		to the performer **of the** request	
		with * *reason for declining* *	

Table E.1: ARS01 (executed by AR01)

when subsidy evaluation **for** [subsidy] **is** declared			(TK02/da)
if	*rightness:*	**the** performer **of the** declaration **is a** subsidy evaluator **the** addressee **of the** declaration **is a** subsidy granter	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	accept	subsidy evaluation **for** [subsidy] **to the** performer **of the** declaration	[TK02/ac]
else	reject	subsidy evaluation **for** [subsidy] **to the** performer **of the** declaration **with** * *reason for rejecting* *	[TK02/rj]

Table E.2: ARS02 (executed by AR01)

when subsidy granting **for** [subsidy] **is** requested			(TK01/rq)
	while	subsidy evaluation **for** [subsidy] **is** accepted	(TK02/ac)
if	*rightness:*	* *no specific condition* *	
	sincerity:	* *no specific condition* *	
	truth:	**the** verdict **of** subsidy **is equal to** 'positive'	
if	*performing the action after* ***then*** *is considered justifiable*		
then	promise	subsidy granting **for** [subsidy] **to the** performer **of the** request	[TK01/pm]
else	decline	subsidy granting **for** [subsidy] **to the** performer **of the** request **with** * *reason for declining* *	[TK01/dc]

Table E.3: ARS03 (executed by AR01)

when subsidy granting **for** [subsidy] **is** promised			(TK01/pm)
if	*rightness:*	**the** performer **of the** promise **is a** subsidy granter **the** addressee **of the** promise **is** *someone related to the subsidy*	
	sincerity:	* *no specific condition* *	
	truth:	* *no specific condition* *	
if	*performing the action after* ***then*** *is considered justifiable*		
then	request	amount determination **for** [subsidy] **to an** amount determiner	[TK03/rq]

Table E.4: ARS04 (executed by AR01)

when amount determination **for** [subsidy] **is** declared			(TK03/da)
if	*rightness:*	**the** performer **of the** declaration **is an** amount determiner	
		the addressee **of the** declaration **is a** subsidy granter	
	sincerity:	** no specific condition **	
	truth:	**the** amount **of** subsidy **is smaller or equal to the** calculated maximum amount **of** subsidy	
if	*performing the action after **then** is considered justifiable*		
then	accept	amount determination **for** [subsidy]	[TK03/ac]
		to the performer **of the** declaration	
else	reject	amount determination **for** [subsidy]	[TK03/rj]
		to the performer **of the** declaration	
		with ** reason for rejecting **	

Table E.5: ARS05 (executed by AR01)

when subsidy granting **for** [subsidy] **is** promised			(TK01/pm)
	while	amount determination **for** [subsidy] **is** accepted	(TK03/ac)
if	*rightness:*	** no specific condition **	
	sincerity:	** no specific condition **	
	truth:	** no specific condition **	
if	*performing the action after **then** is considered justifiable*		
then	request	subsidy payment **for** [subsidy]	[TK04/rq]
		to a subsidy payer	

Table E.6: ARS06 (executed by AR01)

when subsidy payment **for** [subsidy] **is** declared			(TK04/da)
if	*rightness:*	**the** performer **of the** declaration **is a** subsidy payer	
		the addressee **of the** declaration **is a** subsidy granter	
	sincerity:	** no specific condition **	
	truth:	**the** paid amount **of** subsidy **is equal to the** (determined) amount **of** subsidy	
if	*performing the action after **then** is considered justifiable*		
then	accept	subsidy payment **for** [subsidy]	[TK04/ac]
		to the performer **of the** declaration	
else	reject	subsidy payment **for** [subsidy]	[TK04/rj]
		to the performer **of the** declaration	
		with ** reason for rejecting **	

Table E.7: ARS07 (executed by AR01)

when	subsidy granting **for** [subsidy] **is** promised		(TK01/pm)
	while	subsidy payment **for** [subsidy] **is** accepted	(TK04/ac)
if	*rightness:*	**the** performer **of the** promise **is a** subsidy granter **the** ** no specific condition **	
	sincerity:	** no specific condition **	
	truth:	** no specific condition **	
if	*performing the action after **then** is considered justifiable*		
then	execute	subsidy granting **for** [subsidy] **to the** performer **of the** request	[TK01/ex]
	declare	subsidy granting **for** [subsidy] **to the** performer **of the** request	[TK01/da]

Table E.8: ARS08 (executed by AR01)

Curriculum Vitae

Marien Rolin Krouwel

Marien[1] (1986) has been solving (math) puzzles since his early childhood. Around 2000, he built his own PHP-based web CMS that was abstracted from a specific database and applied principles from model interpretation to generate SQL statements. In 2004 Marien started studying Mathematics and Computer Science at Utrecht University. After he had built an application to convert visual programming blocks into executable code for a high-tech customer, Marien received his bachelor's degree in 2008. He decided to continue his Computer Science study at TU Delft, specializing in Information Architecture, a joint track with Policy Management. In 2010 Marien received his master's degree, after he had built a first convertor from DEMO models to a Normalized System in a project together with University of Antwerp and Capgemini. During his studies, Marien taught mathematics and programming to students.

After graduation, Marien started working at Capgemini as business analyst and architect. In twelve years, he held several roles, including software developer, solution architect, people manager, department lead, presales architect, low-code expert, and lead trainer for the architecture portfolio. With more than 10 years of experience in enterprise IT architecture and with cloud, integration, and low-code platforms specifically, Marien supports organizations to apply new technologies in modern (microservice) architectures in order to grasp new business opportunities, by initiating innovative projects and setting up low-code competence centers. He is certified in Mendix, OutSystems, DEMO, TOGAF, ArchiMate, SAFe and O-AA and is certified trainer for Capgemini Academy. Marien regularly provides architecture courses and is an active author and researcher in the field of Enterprise Engineering and Model-driven Software Development. In 2023, he started his own company, focused on Enterprise Model-driven Software Development and low code to quickly support new business ideas with software.

In his spare time, Marien plays the cello, reads detective thrillers and books on leadership. He likes to cycle, practice yoga, and travel to places to surf. Marien is married to Janneke. Together they have 2 children.

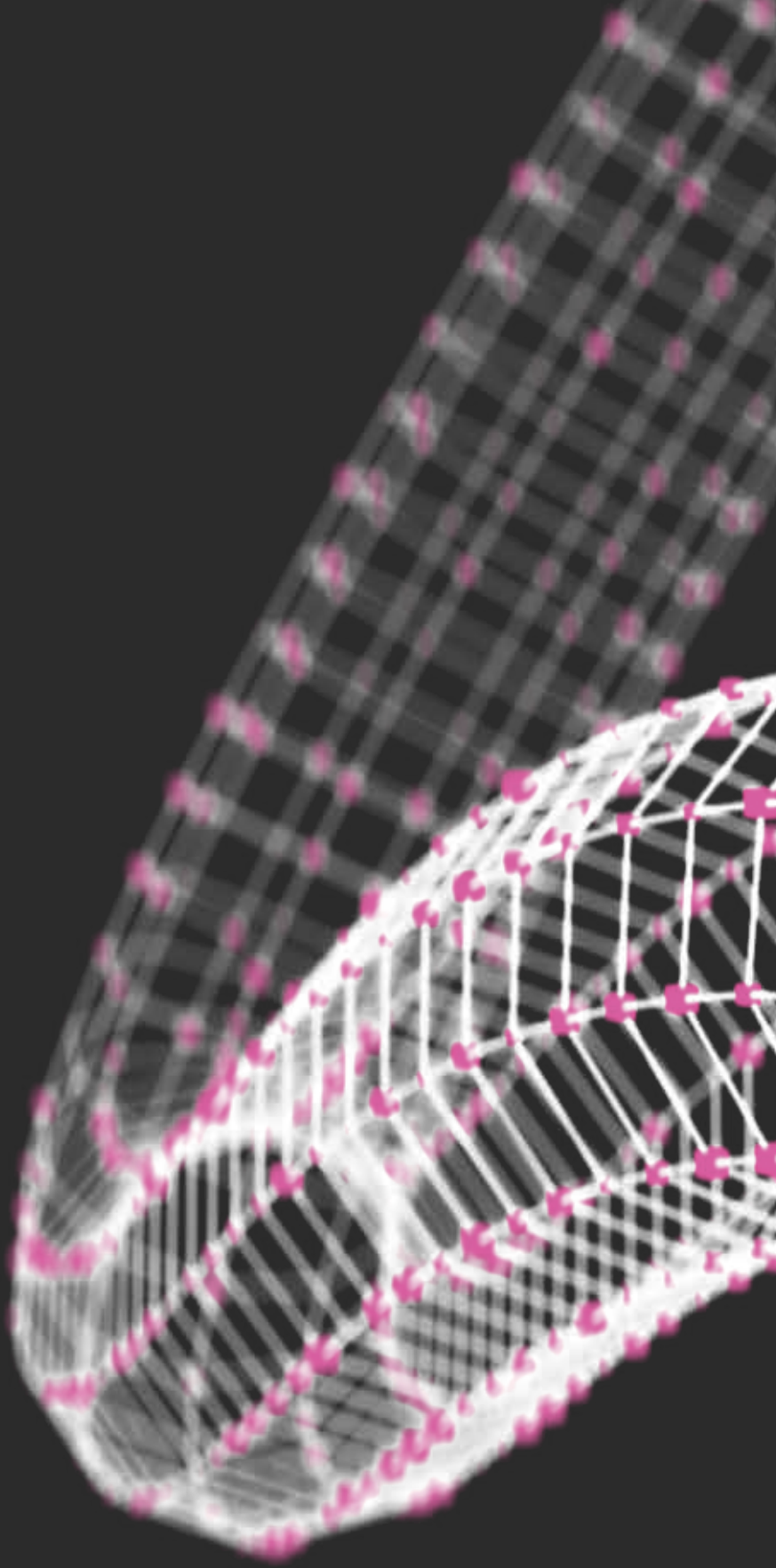
[1] https://www.linkedin.com/in/marienkrouwel/

Colophon

Distributed by:
Marien Krouwel
Make IT Right B.V.
Valeriaanweg 265
3541 TT Utrecht
The Netherlands
marien.krouwel@make-it-right.nl

Cover: Maxim Pashchenko, https://mipashchenko.com/

ISBN: 978-94-6473-242-9



ISBN 978-94-6473-242-9