Software and Systems Modeling (2024) 23:837–864
https://doi.org/10.1007/s10270-024-01156-2

SPECIAL SECTION PAPER

From enterprise models to low-code applications: mapping DEMO to Mendix; illustrated in the social housing domain

Marien R. Krouwel[1,2,3] · Martin Op 't Land[4,5] · Henderik A. Proper[6]

Received: 26 March 2023 / Revised: 18 January 2024 / Accepted: 2 February 2024 / Published online: 8 April 2024
© The Author(s) 2024

Abstract
Due to hyper-competition, technological advancements, regulatory changes, etc, the conditions under which enterprises need to thrive become increasingly turbulent. Consequently, enterprise agility increasingly determines an enterprise's chances for success. As software development often is a limiting factor in achieving enterprise agility, enterprise agility and software adaptability become increasingly intertwined. As a consequence, decisions that regard flexibility should not be left to software developers alone. By taking a Model-driven Software Development (MDSD) approach, starting from DEMO ontological enterprise models and explicit (enterprise) implementation design decisions, the aim of this research is to bridge the gap from enterprise agility to software adaptability, in such a way that software development is no longer a limiting factor in achieving enterprise agility. Low-code technology is a growing market trend that builds on MDSD concepts and claims to offer a high degree of software adaptability. Therefore, as a first step to show the potential benefits to use DEMO ontological enterprise models as a base for MDSD, this research shows the design of a mapping from DEMO models to Mendix for the (automated) creation of a low-code application that also intrinsically accommodates run-time implementation design decisions.

Keywords Enterprise modeling · Enterprise ontology · DEMO · MDSD · Low-code · Mendix

1 Introduction

Due to factors such as extreme (hyper) competition [18, 58, 72], increasing expectations from customers, regulatory changes, as well as technological advancements, the conditions in which enterprises[1] need to thrive become increasingly turbulent. As a result, the ability to change, often referred to as 'agility' [85], becomes an important determinant for the success of enterprises [91, 118]. Indeed, keeping up with competition by reducing the time-to-market of new or changed products is still considered a dominant issue [125].

The notion of 'agile enterprise' is also referred to as the 'run-time adaptive enterprise' [89] or the 'flexible enterprise' [115]. One can, actually, identify different flavors of flexibility that can be considered as cornerstones in creating an agile enterprise, including strategic flexibility, organizational flexibility, financial flexibility, marketing flexibility, manufacturing flexibility and information systems flexibility [115, Fig. 1.4].

Since software makes up a large part of the information systems that are used in modern day enterprises, the enterprise and its supporting software are in fact intrinsically intertwined [27, p. 251]. As a consequence, enterprise agility and software adaptability are also highly intertwined

Communicated by Kurt Sandkuhl, Balbir Barn, Tony Clark, and Souvik Barat.

✉ Henderik A. Proper
Henderik.Proper@tuwien.ac.at

Marien R. Krouwel
Marien.Krouwel@Make-IT-Right.nl

Martin Op 't Land
Martin.OptLand@capgemini.com

1 Make IT Right, Utrecht, The Netherlands

2 Trives Solutions, Utrecht, The Netherlands

3 Maastricht University, Maastricht, The Netherlands

4 Capgemini, Utrecht, The Netherlands

5 Antwerp Management School, Antwerp, Belgium

6 TU Wien, Vienna, Austria

[1] Enterprise, in the context of this research, means any goal-oriented cooperative or network of actors [27], including government agencies, commercial businesses, and non-profit organizations as well as chains of such enterprises.

and can not be viewed separately. Enterprises are almost continuously being redesigned [40], and as a consequence, enterprise and information systems engineering, including software development, is increasingly an ongoing activity that requires continuous alignment between the enterprise and its supporting software [51].

One of the issues in the alignment of an enterprise and its supporting software, is that design decisions that concern an enterprise's implementation, including functionary types, organizational units, and the choice to apply the four-eyes principle, are often hard-coded into software [10, 47, 124]. The latter makes it harder to align changes on the enterprise level with changes in the software—something the authors recognize in their current practice quite frequently.[2] Moreover, the authors recognize that the software often does not support the *required* enterprise agility, but instead offers more flexibility than required on some aspects and less than required on (critical) other aspects. In order for software to provide the flexibility as required by the enterprise, it is necessary to make enterprise implementation design decisions explicit and to make transparent how they are implemented in software.

In order to prevent developers from introducing implicit design decisions in the process of software development, and to ensure that the software supports the level of flexibility as required by the enterprise, MDSD is suggested [30]. MDSD is a model-driven approach for the development of software that promises to bridge the gap between requirements of an enterprise and software implementation [12, 92, 113]. It uses models and model transformations to generate code or for real-time interpretation by running software [8, 12, 33, 48, 77, 106]. Metamodels play a crucial role in defining a model transformation or mapping function [2, 6, 8, 57] (see Fig. 1) and need to be precisely defined [12, 63, 77].

One of the earliest approaches toward MDSD is Computer-aided Software Engineering (CASE) [35, 71, 73] from the 1980s. A more recent approach is Model-driven Architecture (MDA),[3] that relies heavily on the use of Unified Modeling Language (UML) models [105]. 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 [81, 110]. Other approaches include the automatic transformation of UML or Business Process Modeling and Notation (BPMN) models into BPEL [90, 126] specifications. While both modeling languages have a defined metamodel, it has also been shown that these lack proper semantics or [14, 29, 36, 64], even worse, completely seem to miss proper concepts for enterprise modeling [128].

[2] Two of the authors are practitioners, with an average of 25 years of experience in the field of enterprise (architecture) modeling and (model-based and model-driven) software development.

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

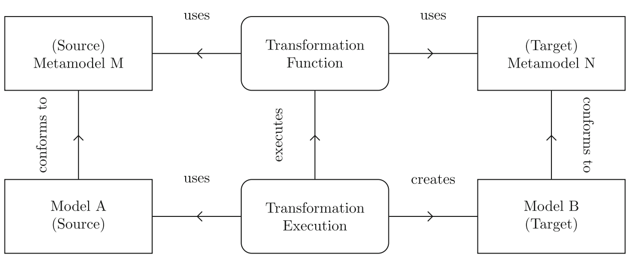


Fig. 1 Model transformation and the role of metamodels, adapted from [7, 8, 12, 17]

In this research, the general hypothesis is that by taking an MDSD approach one can improve enterprise agility, at least to the point that software development is not the limiting factor for enterprise agility anymore. As a more specific hypothesis, we posit that DEMO ontological enterprise models—capturing the enterprise's essence—, supplemented with explicit enterprise implementation design decisions, captured in Organization Implementation Variables (OIVs), provide a semantically rich starting point for MDSD.

In terms of Fig. 1, this means we will use DEMO as the source for MDSD. As a first step to, indeed, show the potential benefits to use DEMO ontological enterprise models as a base for MDSD, this research shows the design of a mapping from DEMO models to Mendix for the (automated) creation of a low-code application that also intrinsically accommodates run-time implementation design decisions. So, in terms of Fig. 1, we will use Mendix as the target for MDSD. In the next section, we will provide a further motivation for this source and target.

In line with this reasoning, the overall goal of the research reported on in this article is to create an (automated) transformation, a *mapping*, from the model of the enterprise essence and associated OIVs to a Mendix application in order to be able to generate low-code applications from enterprise models. As this mapping is a design(ed) artifact, the Design Science Research methodology [50, 76, 109, 123] has been applied. In terms of Design Science Research (DSR), the main focus of this article is on the design cycle, i.e., the construction and (initial) evaluation of the mapping. The resulting mapping has been evaluated (and further improved) multiple times, involving different input models, ranging from academic cases such as EU-Rent [83, 88] to real-world cases including Social Housing [66, 89]. While the mapping has evolved and improved over time, the latest version of the mapping is presented.

The remainder of this article is structured as follows. In Sect. 2 related work such as MDA is elaborated on and the rationale for the chosen source and target for the MDSD approach are provided. Given these choices, Sect. 3 outlines the theoretical background regarding Enterprise Ontology, Enterprise Implementation, and low-code technology. In Sect. 4 the mapping is shown, while Sect. 5 provides a summary of the evaluations conducted so-far and shows how the evaluation was done on the real-world Social Housing case. Before concluding, Sect. 6 discusses the limitations of the mapping. This article ends with conclusions and directions for future research in Sect. 7.

2 Related work and chosen solution direction

In this section, we discuss related work that is of direct relevance to the ambition to create an (automated) transformation from the DEMO model of the enterprise–its 'essence'–and associated implementation design decisions to a working Mendix application in order to be able to generate low-code applications from enterprise models.

2.1 Enterprise engineering

The field of Enterprise Engineering sees enterprises as complex sociotechnical systems [40] and therefore connects the fields of organizational science and software development. DEMO (Design and Engineering Methodology for Organizations) is a leading method within the discipline of enterprise engineering [26], with strong methodological and theoretical roots [3, 46, 104, 122] that sets communication as the primal notion for the design of enterprises and its supporting software systems [95]. At the same time, we observe there to be an increasing uptake of DEMO in practice, as, e.g., illustrated by the active community and certification institute[4] as well as the reported cases concerning the use of DEMO [4, 10, 19, 20, 27, 32, 39, 54, 55, 60, 68, 79, 86, 87, 94, 107] and integration with other mainstream enterprise modeling approaches such as ArchiMate [31, 62] and BPMN [16, 42, 52, 78, 93, 120].

The theories behind DEMO distinguish between *Enterprise Ontology* and *Enterprise Implementation* [27]. Enterprise Ontology as defined by Dietz aims to capture the 'essence' of an enterprise in terms of products and services delivered. The Enterprise Implementation pertains to the organizational structure(s) and (sociotechnical) means, including the division of work between human actors and software solutions, as well as their assignment(s) to the organizational structures. Although changes to an enterprise's ontology do occur, most of the time changes pertain to its implementation only [25, 27, 95]. As mentioned above, the authors have observed in practice how such enterprise implementation design decisions are often hard-coded into software.

[4] https://ee-institute.org.

2.2 Model-driven software development and MDA

Claimed advantages of MDSD over traditional software development approaches include: (a) a common and better understanding and reasoning about the required or created system [6, 12, 48], (b) the possibility to simulate before building the system [48, 63], (c) an increased productivity of the development team due to (partial) automation of the development process [6, 12, 48, 63], (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 [12, 48, 63], and (e) traceability between model and code [1, 102]. It is because of all of these advantages that in the research as reported on in this article, an MDSD approach is applied.

A specific implementation of MDSD is MDA, that, originally,[5] discerns three layers or kinds of models being used [82][6]:

1. The Computation Independent Model (CIM) describes the business domain without any computational aspect;
2. The Platform Independent Model (PIM) focuses on the software that supports the business but is technology independent;
3. The Platform Specific Model (PSM) is very closely related to the PIM but adds technological details so that working software can be created.

In terms of MDA this research thus needs an enterprise modeling language that can act as CIM and a target technology that can act as PSM ultimately. Considering the PIM and PSM are very closely related, in this research the distinction between the two is considered not very relevant at this stage.

2.3 Source model: enterprise modeling language

In order to be able to transform a model in the process of MDSD (see Fig. 1), the input model must a) be comprehensive, b) describe all business requirements, c) be consistent, and d) have its semantics fully specified. As the aim of this research is to create enterprise software, the chosen source models should at least comprise actor roles, products, process, information (items) and business rules. There are many enterprise modeling languages, including for example: 4EM [70, 101], ArchiMate [69, 117], ARIS [103], BPMN,[7]

[5] In version 2.0 [84] the terms can still be found but they are not defined as elaborate as in version 1.0.

[6] A more elaborate introduction to these layers can be found in [59].

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

CogNIAM,[8] MEMO [37], SBVR,[9] SYSML[10] and UML.[11] However, for each of these modeling languages it holds that either a) it describes only a part of the enterprise, e.g., it focuses on process or data or business rules; or b) it lacks formal semantics [65].

DEMO ontological enterprise models are posited as being coherent, comprehensive, consistent and concise models of the (essence of the) enterprise [27, p. 14]. Such a model is coherent of sub models describing both products, actor roles, processes, information and business rules. Moreover, the metamodel and semantics of DEMO is fully described [24]. As such, they seem to provide a good starting point for an MDSD approach. This is backed up by experiences from practice that indicate that DEMO ontological models provide a good basis to define requirements and design or generate software [19, 27, 32, 39, 54, 55, 60, 79].

Alternatively, a combination of several modeling languages could be chosen for the purpose of this research. However, given the lack of proper semantics for most existing modeling languages, combining languages is a tedious task that goes beyond the timelines of this research. As DEMO is readily available and seems to fit the goal of this research, DEMO models will be used as source for the MDSD approach as applied in this research.

2.4 DEMO-driven software development

While a literature study resulted in some partial mappings from DEMO to working software [53, 56, 61, 111], a complete mapping from the full DEMO metamodel to any software implementation model was not found. Furthermore, just as important, existing research seems to neglect enterprise implementation aspects. This results in a need for software developers to deal with these aspects themselves, thereby (possibly) causing a lack of transparency regarding the transformation of implementation design decision to software; and thus (possibly) hampering enterprise agility. For instance, Van Kervel et al. [60, 61, 111] report on a software engine that is able to generate working operational software, taking the DEMO ontological enterprise model of a domain as input. A deeper investigation, however, revealed that, in the process of creating this engine, several implicit implementation design decisions have been made that ended up being hard-coded in the software engine. Moreover, this engine only seems to support changes on the level of the DEMO ontological model of the enterprise, 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.

A potential explanation for these issues is the fact 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 researchers suggest using a model-driven code generation strategy, since real-time interpreters can become even more complex. At the same time, such an approach makes it harder to make (controlled) customizations and extensions that are often needed in practice, and it is therefore the researchers suggest to turn to low-code technology.

2.5 Target model: low code

The idea of low-code technology is to add an(other) abstraction layer on top of (high) code such as Java and.Net. In terms of MDA [84], it is actually not clear if low code should be regarded as a PIM or as a PSM.

Low-code technology applies techniques from MDSD, while still allowing for (controlled) customizations and extensions [15]. While low-code technology improves enterprise agility, compared to traditional code [108], practical experience shows that these platforms mainly offer technical adaptability. For instance, changing the database management platform from MySQL to Postgres, and some flexibility on the (functional) requirements level, such as changing a specific workflow or screen lay-out. At the same time, however, changes in the enterprise, such as the organizational structures or its portfolio products and services, can still take quite some time to implement.

For this research, the Mendix low-code platform[12] is chosen because it provides good documentation about its metamodel[13] and offers a Software Development Kit (SDK) to create Mendix applications using TypeScript.[14] Next to that, the researchers are experienced in using the Mendix platform in real-world situations.

3 Theoretical background

Before discussing the mapping from DEMO to Mendix, this section will briefly introduce the most important aspects of Enterprise Ontology and Enterprise Implementation, as well as the concept of low-code technology for the software support of enterprises.

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

[9] https://www.omg.org/spec/SBVR/.

[10] https://sysml.org/.

[11] http://www.uml.org/.

[12] https://mendix.com/.

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

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

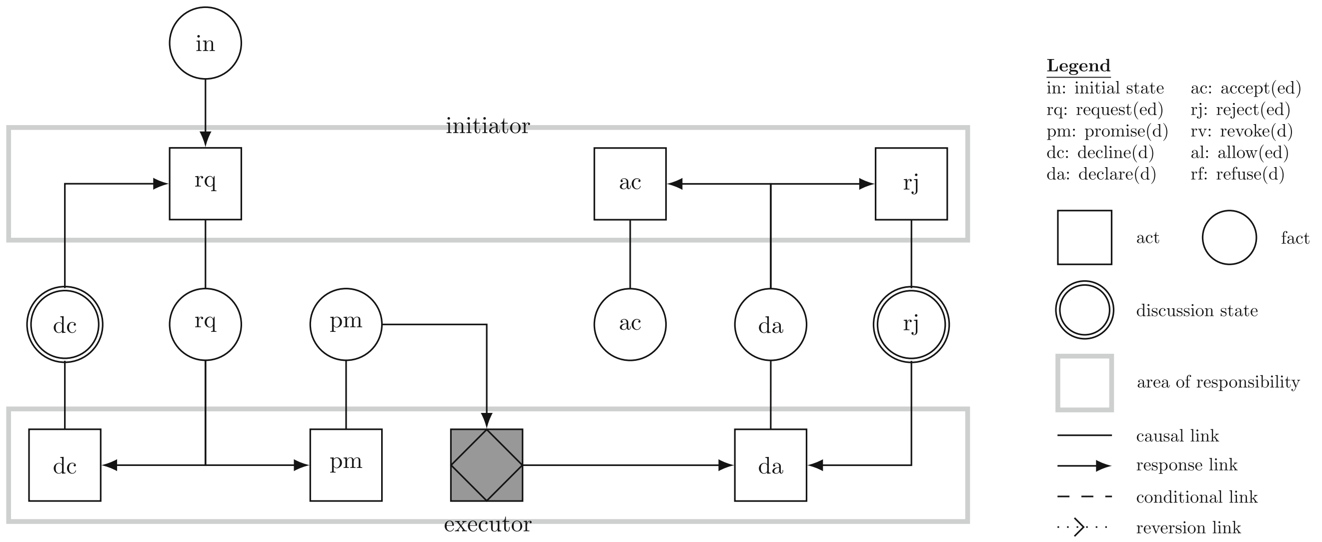


Fig. 2 The standard transaction pattern shows the basic flow and discussion states. The complete transaction pattern includes 4 revocation patterns [27]

Table 1 Transactor Product Table (TPT) for the Volley case

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

3.1 Enterprise ontology

In general, an ontology is defined as a formal and explicit conceptual specification describing (shared) knowledge about some domain [11, 43, 45, 114]. In contrast to, e.g., 'regular' (data) models, an ontology specifies a shared view of the involved stakeholders [112] that can be expressed in a model containing the relevant concepts including their definition or semantics, and their relations [34, 44, 112].

Enterprise Ontology involves a specialization of the general concept of ontology, in that the goal of Enterprise Ontology is to share knowledge about an enterprise within and between enterprises [119]. Building upon the Language/Action Perspective [122], Fox [34] proposes actors, their roles and communication links between them as key concepts in Enterprise Ontology.

In his EE-theories for Enterprise Engineering and Enterprise Ontology, Dietz further builds upon these concepts by seeing an enterprise as a network of actors that enter into and comply with commitments [27]. Such commitments are raised by actors in acts, the atomic units of action, and follow a generic pattern called the Complete Transaction Pattern (CTP). This CTP (see Fig. 2) consists of 19 *general step kinds* and deals with the basic flow—request, promise, execute, declare and accept—as well as discussion states—decline, reject—and cancellations (or revocations). The general working principle is that actors constantly check whether there are acts they have to deal with or respond to; this is the so-called *actor cycle*. The total set of acts for an actor to deal with is called the actor's *agenda*.

By abstracting actors to *actor roles* and commitments regarding a specific *product kind* to *transaction kinds*, the (DEMO) model becomes independent of the actors that are actually involved in the operation. Every elementary transaction kind has exactly one actor role as executor and one or more actor role(s) as initiator. This abstraction results in an enterprise model that only depends on an enterprise's products and services, and that is fully independent of the way in which an enterprise is realized and implemented [22]. As a result, the DEMO ontological enterprise models are considered more stable than implementation dependent models [25]. In the next sections, the DEMO aspect or sub models, along with an example, and its metamodel, will be introduced.

3.1.1 Ontological aspect models

The DEMO ontological model of an enterprise consists of an integrated whole of four aspect models [27]. The *Cooperation Model (CM)* models the *cooperation* of the enterprise; it consists of transaction kinds, associated (initiating and executing) actor roles, fact banks, 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 TPT. The *Process Model (PM)* models the *processes* that take place as the effect of acts by actors, by detailing the coordination between actor roles; it makes explicit the causal and wait links between acts from the CTP. The PM is expressed in one or more Process Structure Diagrams (PSDs) and one or more Transaction Pattern Diagrams (TPDs). The *Fact Model (FM)* is the semantic model of *products* of the enterprise; it defines (declared or derived) fact types (entity types with their related product kinds, property types, attribute types and value types), existence laws and occurrence laws. The FM is expressed in an Object Fact Diagram (OFD) and zero or more Derived Fact Specifications (DFSs). The Action Model (AM) is the model of the *operation* of the enterprise, guiding actors in performing their acts. It specifies for every agendum kind with which the enterprise has to deal one or more Action Rule Specifications (ARSs). Each ARS fully supports the actor cycle and consists of an *event part* detailing the agendum kind to respond to, an *assess part* detailing the conditions to check and a *response part* that states how the actor should respond. Although these guidelines may look like rules, they offer the actors the possibility to autonomously–but responsibly–deviate from these guidelines.

3.1.2 Example: Volley

We use the Volley case [24, Ch. 12], which basically represents any (amateur) sports club, as an example to illustrate some of the key elements of DEMO.

In this case one can become member (TK01 for the first Transaction Kind) of the sports club by filling in a form with some personal details. After the information has been checked, the president of the club decides whether a membership is granted or not–the membership is, usually, granted when the (aspirant) member is 18 years old or more. When it is decided that the aspirant member can indeed become a member, a request for the payment of the first period (or part thereof) is done (TK02). As soon as the payment has been made, the membership card is printed and sent to the member so that he or she can start using the sports facilities. Needless to say, after becoming a member, a periodic process will make sure the periodic payments are being made. Moreover, there are ways to end a membership. However, in this example the focus is on becoming a member, and other parts are left out for simplicity.

The TPT for the Volley case (Table 1) shows the (two) identified (elementary) transaction kinds and for each transaction kind its related product kind and the (elementary) actor role that is the executor of the transaction kind. For simplicity, PK01 only allows for a one-time payment per membership; in order to allow for periodic payments, it has to be changed to '[membership] for [period] has been paid' with consequences for the FM. The CSD for the Volley case (Fig. 3)

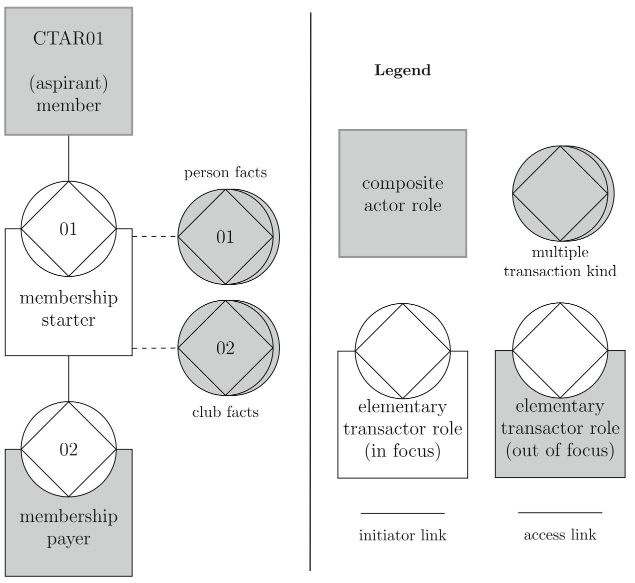


Fig. 3 CSD for the Volley case

visualizes the TPT and shows the actor role(s) that can act as the initiator of a transaction kind; CTAR01 is a composite actor role (gray background with large border) and AR02 is out of focus (gray background), implying that only for TAR01 ARSs have to be created. It also shows there are two external fact banks (multiple transaction kinds) that contain facts about persons, e.g., their day of birth, and about the club, e.g., the minimum age. As shown earlier, each transaction kind follows the CTP, consisting of 19 general step kinds, resulting in 19 transaction kind step kinds for each transaction kind.

The PSD for the Volley case (Fig. 4) shows the dependencies between the two transaction kinds. It tells that after a promise for TK01, (usually) a request for TK02 is performed (causal link), and that an execute for TK01 has to wait until the related TK02 has been accepted (wait link). The dependencies will be formalized in the AM.

The OFD for the Volley case (Fig. 5) shows the primary entity type 'MEMBERSHIP', its attribute type 'starting day' and its property types 'member' and 'payer'. It also shows the entity type 'PERSON' with attribute type 'day of birth' and entity type 'YEAR' with attribute types 'minimum age' and 'membership fee', implying these club facts can be adjusted yearly. Both entity types PERSON and YEAR are out of focus (gray background). The DFS for the Volley case (Table 2) shows the derivation rule to calculate the age for a given person on a given day.

ARS01 (Table 3) for TAR01 in the Volley case shows the action rule to deal with a request of TK01:

- The *event* part shows the event that is being dealt with (TK01 is requested), including the relevant information

Table 2 Transactor Product Table (TPT) for the Volley case

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

Table 3 ARS01 (executed by TAR01) for the Volley case

event	**when** membership starting **for** [membership] **is** requested			(TK01/rq)
		with	**the** starting day **of** [membership] **is some** day	
			the member of [membership] **is some** person	
			the payer **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** starting day **of** [membership] **is greater than or equal to the** minimum age **of the** year **of the** starting day **of** [membership]	
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 **	

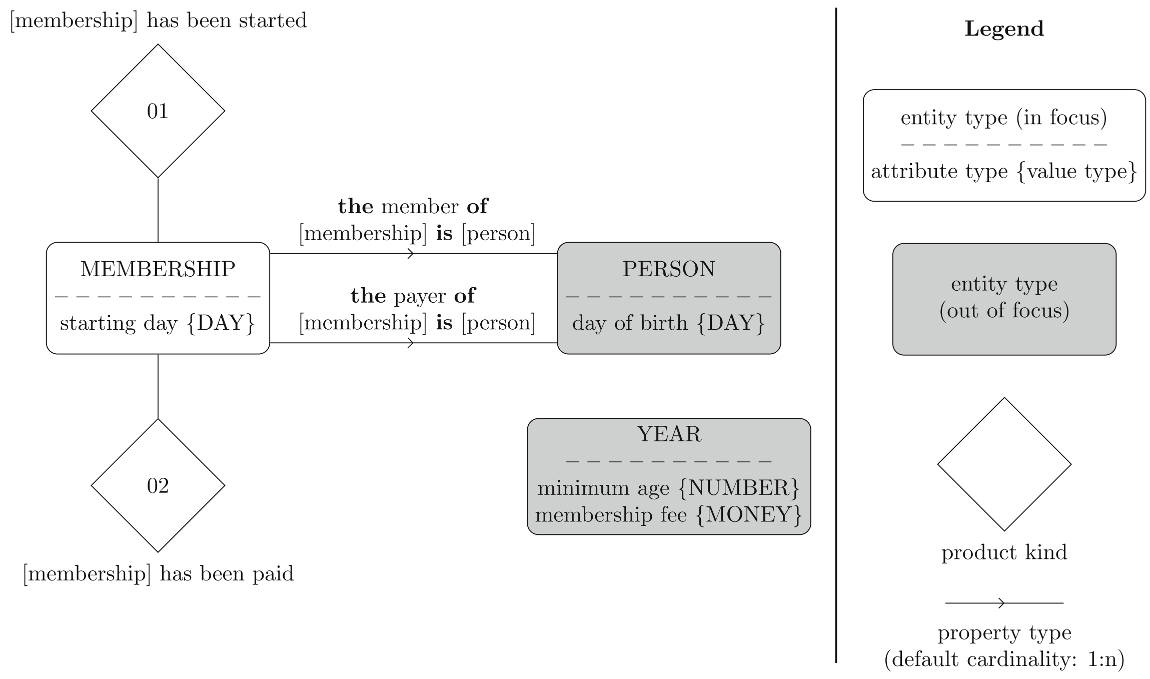

[membership] has been started
01
the member of [membership] is [person]
the payer of [membership] is [person]
MEMBERSHIP
starting day {DAY}
PERSON
day of birth {DAY}
YEAR
minimum age {NUMBER}
membership fee {MONEY}
02
[membership] has been paid
Legend
entity type (in focus)
attribute type {value type}
entity type (out of focus)
product kind
property type (default cardinality: 1:n)

Fig. 4 OFD for the Volley case

that must be part of the request in order to be able to deal with it.

– The *assess* part shows the three clauses after Habermas [46]:

  – In the rightness division, it is ensured that both involved actors have the proper authority. In this case, the person performing the request should be the member of the (requested) membership.

– In the sincerity division, it is checked whether both actors are trustworthy and sincere in their commitment. In this case, no specific condition has to be checked.

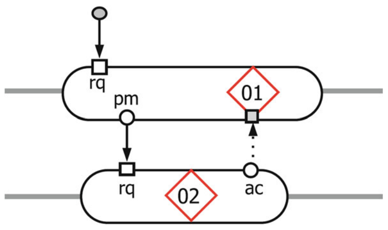


Fig. 5 PSD for the Volley case

  - In the truth division, possible violation of existence and occurrence rules are checked. In this case, the age of the (aspirant) member is checked against the minimum age.
- The *response* part shows that the actor can choose between promising and declining the request, based on the assessment. The first sentence of the response part allows the actor to responsibly (!) deviate from the rule. For example, if a young boy named Federer wants to join the club, but has not reached the minimum age yet, the actor can decide to allow the boy to become member. This underpins the idea that actors act autonomously and responsibly.

In total, there are 4 ARSs defined for TAR01. Together they constitute the AM for the Volley case.

3.1.3 DEMO metamodel

The DEMO Specification Language (DEMO-SL) specification [24] provides the formal specification of the DEMO metamodel, which has been extended and improved further in [80]. In Fig. 6, the DEMO metamodel derived from DEMO-SL is depicted. The general step kinds are defined by the CTP 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 the CM, PM, and FM, but no (detailed) action rules are created.

While there is an XML-based exchange model for DEMO available [80], in this research it was decided to create a more compact JSON (JavaScript Object Notation) format to represent the DEMO models as input for the automated model transformation, leaving out parts that are for representation only. The DEMO model for the Volley case expressed in JavaScript Object Notation (JSON) is shown in Fig. 7. The (metamodel) concepts of transactor role, elementary actor role, elementary transaction kind and product are compressed into one JSON object called transactor role, where an ID and names are added for readability. The composite actor roles and multiple transaction kinds can be found as such in the JSON representation, along with the initiates and has access links between an (elementary or composite) actor role and a (multiple) transaction kind. The concept of transaction kind step kind can be fully derived from the transaction kinds and the CTP, and thus do not need to be specified in the JSON input, just as causal links and wait links, which are (redundantly) specified in the ARSs as well. The different specializations of fact kind are represented with a type attribute (entity type, property type, value type, attribute type, derived), with a name and, if applicable, a range and domain. The ARSs are represented by an identifier, the actor role responsible for the rule, the event part (consisting of a when and while clause) and the assess part (consisting of a then and, optionally, else clause).

3.2 Enterprise implementation

In order for an enterprise to become and stay operational, it has to be implemented with appropriate technologies [27], including the assignment of work to human actors and/or IT-based actors. While the DEMO ontological enterprise model shows the (decided) products and services (as transaction kinds) as well as the collaboration network (of actor roles) for the production and delivery, many degrees of freedom exist on how to operationalize this network toward a ‘running’ *implemented* enterprise [27, p. 46].

Implementation design starts from an enterprise’s ontological model and ends with a fully detailed specification of the implementation of the enterprise—possibly within the design space constrained by an Enterprise Architecture [23]. The lowest level and most detailed model of an enterprise describes all implementation design decisions, including organizational structure(s), the decision to automate tasks, as well as the assignment of people and means to the organizational structures [67].

The notion of OIV expresses a category of design decision(s) for enterprise implementation [67]. It is a variable in the sense that its value can be different for different implementation alternatives. Some examples of the ∼30 of such OIVs [67] include:

- 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), since it is common practice to authorize persons to fulfill func-

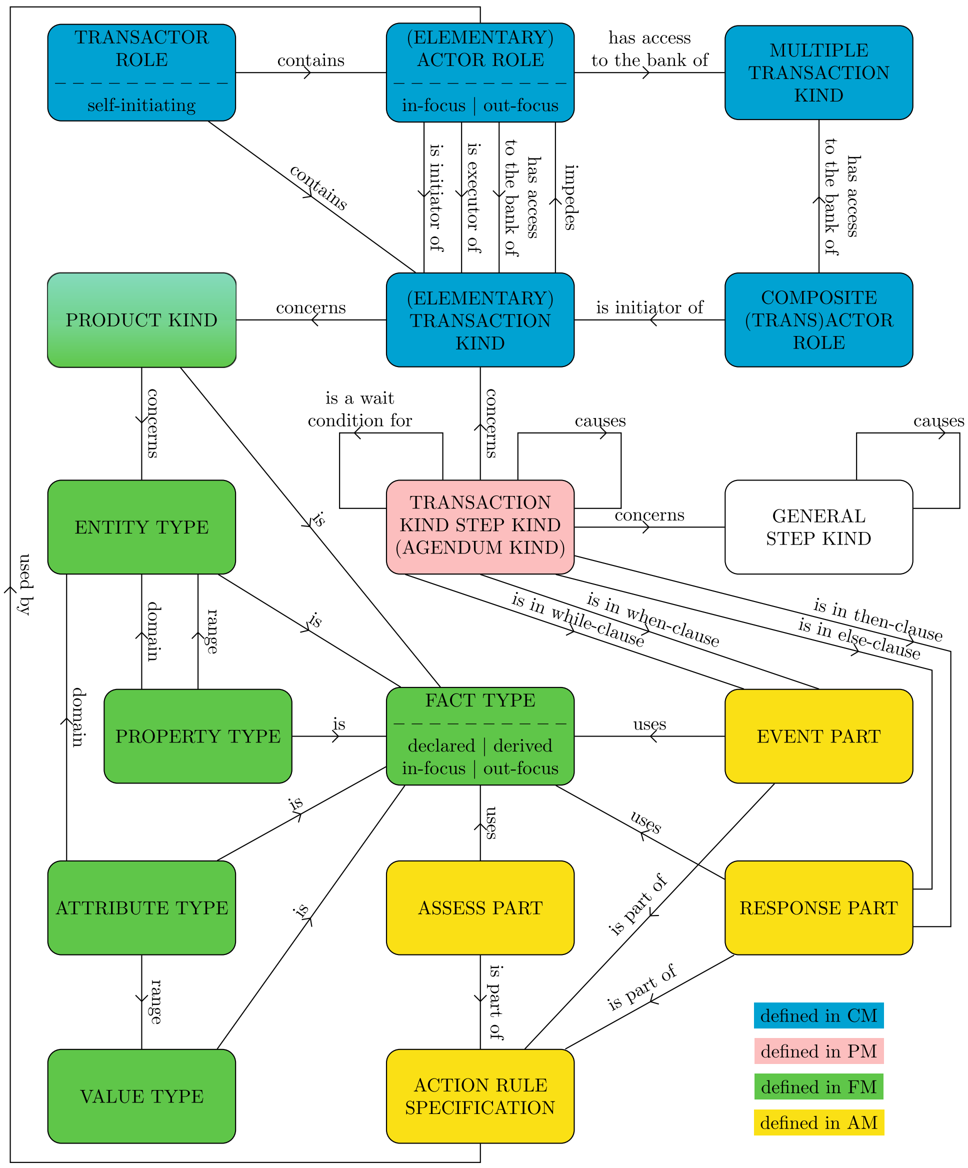


Fig. 6 DEMO metamodel, adapted from [24, 80]. Legend is same as for FM, see Fig. 4

tionary types and not actor roles directly—e.g., the functionary type ‘cook’ that fulfills both the actor roles ‘order baker’ and ‘stock controller’; or the functionary type ‘deliverer’ who is authorized for the acts ‘promise’ and ‘declare’ of the transaction kind ‘order delivery’ and also for the act of ‘accept’ for the ‘order payment’ transaction kind; and

– Deciding on the *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 afterward (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.

The implementation of an enterprise, and therefore the specific value chosen for the OIVs, will change far more often than the products and services it delivers. For example, the actor roles ‘order completer’ and ‘order baker’ are stable notions in a pizzeria, just as the act ‘accept order payment’; what might change often is the authority of a functionary type—answering questions such as ‘*should we combine the actor roles of order completer and order baker into one functionary type in the first months of opening our new pizzeria branch, while later we might distribute these responsibilities over the functionary types ‘sales’ and ‘baker’?*’ or ‘*should we take the responsibility for accepting order payments away from our functionary type deliverer and move it to an automated web agent under the (outsourced) responsibility of a payment service provider?*’.

```
{
    "transactorroles": [
        {"id": "01", "transactionkindname": "membership starting", "actorrolename": "membership starter",
         "focus": true, "casekinds": ["Membership"], "product": "[membership] is started"},
        {"id": "02", "transactionkindname": "membership paying", "actorrolename": "membership payer",
         "focus": false, "casekinds": ["Membership"], "product": "[membership] is paid"}
    ],
    "compositeactorroles": [
        {"id": "01", "name": "(aspirant) member"}
    ],
    "multipletransactionkinds" : [
        {"id" : "01", "name" : "person facts"},
        {"id" : "02", "name" : "club facts"}
    ],
    "initiates" : [
        {"ctar": "01", "tk": "01"},
        {"ear": "01", "tk": "02"}
    ],
    "hasaccess" : [
        {"ear": "01", "mtk": "01"},
        {"ear": "01", "mtk": "02"}
    ],
    "factkinds": [
        {"name": "Membership", "type": "entitytype", "focus": true},
        {"name": "Person", "type": "entitytype", "focus": false},
        {"name": "Year", "type": "entitytype", "focus": false},
        {"name": "Year", "type": "valuetype"},
        {"name": "Day", "type": "valuetype"},
        {"name": "Money", "type": "valuetype"},
        {"name": "Number", "type": "valuetype"},
        {"name": "member", "type": "propertytype", "domain": "Membership", "range": "Person"},
        {"name": "payer", "type": "propertytype", "domain": "Membership", "range": "Person"},
        {"name": "starting day", "type": "attributetype", "domain": "Membership", "range": "Day"},
        {"name": "day of birth", "type": "attributetype", "domain": "Person", "range": "Day"},
        {"name": "year", "type": "attributetype", "domain": "Year", "range": "Year"},
        {"name": "minimum age", "type": "attributetype", "domain": "Year", "range": "Number"},
        {"name": "membership fee", "type": "attributetype", "domain": "Year", "range": "Money"},
        {"name": "PersonAge", "type": "derived", "domain": ["Person", "Day"], "range": "Number"}
    ],
    "actionrules": [
        {"id": "01", "ear": "01", "when": "01-rq", "while": [], "then": ["01-pm"], "else": ["01-dc"]},
        {"id": "02", "ear": "01", "when": "01-pm", "while": [], "then": ["02-rq"]},
        {"id": "03", "ear": "01", "when": "02-da", "while": [], "then": ["02-ac"], "else": ["01-rj"]},
        {"id": "04", "ear": "01", "when": "01-pm", "while": ["02-ac"], "then": ["01-ex", "01-da"]}
    ]
}
```

Fig. 7 JSON file representing the DEMO model for the Volley case

For an agile enterprise it is priority that frequently occurring changes, typically implementation design decisions, are not on its critical path. Ideally it should be possible to make such changes with no or only little impact in the supporting software. 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. For instance, departments may be rearranged yearly, while work locations may be more stable. OIVs offer the possibility to make explicit the enterprise's implementation design decisions that need to get a place in software and/or to make an explicit requirement that (some) values should be (easily) changeable in the software.

At the same time, the variability can be quite high, making it difficult to gain overview and impossible to support all implementation possibilities. With the simplistic assumption that each of the 14 (out of the 30) OIVs that can be changed independently can have three different values each, the total number of possible implementation alternatives is already $3^{14} \approx 4.8 * 10^6$ for each transaction kind—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.

3.3 Low-code technology

The term 'low code' was first coined by Forrester in 2014 [96]. Although some say it is neither clear what low code exactly is, nor that its features are very new [9, 15], low-code development platforms claim to enable creation of software with less effort compared to traditional[15] programming [121]. It builds upon existing concepts including MDSD, code generation and visual programming [21]. Claimed benefits of low-code technology include [49, 74, 98, 100, 108] a) less hand-coding, faster development, and, as a result, cost reduction in both development and maintenance, as well as a shorter time-to-market; b) complexity reduction by using pre-built 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.

[15] Also known as 'high code', such as Java and.Net.

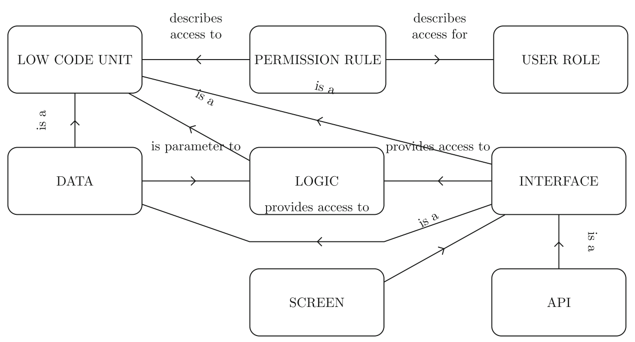


Fig. 8 Low-code metamodel

While low-code platforms support technical flexibility such as changing from one database to another, they do not prevent developers from hard coding (implicit) implementation design decisions [10]. Other mentioned disadvantages of low-code technology include poor scalability, performance, vendor lock-in and limited creativity and customizability [108]. There is however also research that argues against these disadvantages or that shows that the practical implications of these disadvantages is non-existent [38, 49, 97, 127].

Low-code platforms, including Mendix, rely on three central concepts (see Fig. 8): a) *data*, b) *logic* (also called action or (micro)service), and c) and *interface* (Application Programming Interface (API) and 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 set of low-code concepts is richer and closer to the business, making it easier to, e.g., connect the information need of a user to an interface to retrieve and show the corresponding data. The Mendix metamodel is a specialization of the generic low-code concepts (see Table 4).

4 Mapping

The mapping is based on the DEMO metamodel as provided, extended with the concept of Organization Implementation Variable, and can be found in Table 5. The result of applying the mapping function to a DEMO model is a full-functional Mendix app containing a data model (and database), (basic) CRUD (Create, Read, Update, and Delete) 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 function to other applications. In the process of creating the mapping, several design decisions have been made that are reported below.

D1 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 D4.

D2 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 D5.

D3 As the page for showing the agenda for an actor is a very generic functionality, it was decided not to generate it but to built it 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 article as they merely build upon earlier research [41, 60, 61, 111].
The development of this module has started before Mendix launched its native workflow capabilities, that has the potential to reduce complexity of this module.

D4 $Transaction.Proposition$ is an Entity that is part of the generic module handling the CTP. By extending it, the generic state machine can be used, but it can also be related to the specific entity or entities the Product Kind is about, i.e., the variables in the Product Kind.

D5 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'.[16] As this functionality does not seem available (yet) through the Mendix SDK, it was decided to not use that it. Instead, some basic CRUD pages were created in order to view and modify the data. It is fairly easy in the generated application to change this later.

D6 There were not enough example DEMO models to provide a mapping for the generalization and specialization Entity Types.

D7 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

[16] Mendix Connect is a collection of functionalities 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/.

Table 4 Mendix (metamodel) units as specialization of the generic low-code concepts

Low-code concept	Mendix unit
Data	Entity with attributes and associations, enumeration
Logic	Microflow
Screen	Page, typically containing a data grid or view and (Action) Buttons
API	Published (web or REST) service
User role	User role
Permission rule	Access rule

Table 5 Mapping from DEMO metamodel (concept) to Mendix metamodel (unit)

DEMO Concept (Aspect Model)	Example	Mendix Unit
Elementary Transaction Kind (CM)	TK01	n/a, see D1
Multiple Transaction Kind (CM)	MTK01	n/a, see D2
Actor Role (both elementary and composite) (CM)	AR01	User Role, see D3
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 D4
Transaction Kind Step Kind (PM)	TK01/rq	Page
Declared Entity Type (FM)	Registration	Entity and Pages, see D5
Aggregation Entity Type (FM)	{Registration X Year}	Entity with Associations to its aggregates
Specialization Entity Type (FM)	Started Registration	n/a, see D6
Generalization Entity Type (FM)	n/a	n/a, see D6
Property Type (FM)	member	Association
Declared Attribute Type (FM)	starting day	Attribute or Association, see D7
Calculated Attribute Type (FM)	age	Microflow, see D8
Value Type (FM)	day, money	Enumeration or Entity, see D7
Action Rule-Event Part (AM)		Action Button and (Microflow) Access Rule, see D9
Action Rule-Assess Part (AM)		Microflow and Page, see D9
Action Rule-Response Part (AM)		Action Button, Microflow and (Microflow) Access Rule, see D9
Organization Implementation Variable	Functionary Type	Entity, see D10

Mendix (primitive) AttributeType, e.g., DateTime, Decimal, etc. As this mapping is purely a matter of (software) implementation, it is put into a separate descriptor file.

D8 Calculated Attribute Types need to be calculated, for which currently a so-called Microflow is created. The (mathematical) definition of the calculation (as defined in a Derived Fact Specification) needs to be implemented in that Microflow, of which the mapping is too extensive and detailed for the scope of this article. A decision that goes along with this choice is that from a performance perspective one would like to be able to decide whether this calculation is performed on read or on save. The low-code approach applied in this research makes it easy to make such a decision in the platform, as it currently seems too difficult to include this aspect into the mapping.

D9 For the handling of Action Rules, a Mendix Action Button for the User Role that has to deal with the agendum (kind), 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 are generated. 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 mapping of the assess part is similar to that of a Calculated Attribute Type and thus a similar reasoning holds as described in D8.

D10 In this research the principle is adopted that 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

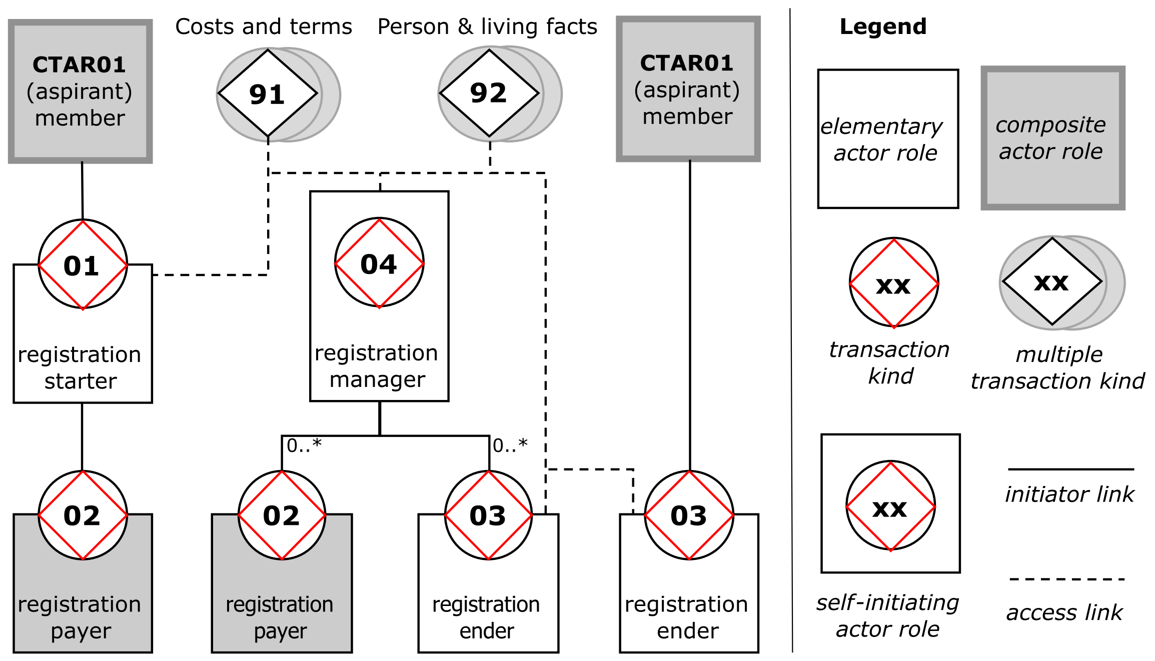


Fig. 9 CSD for social housing

Table 6 TPT 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

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 technology as target platform because it easier to build this into the state machine than into the mapping. At the same time, in Sect. 3.2 it has been shown that the possible number of configurations grows exponentially with the number of OIVs and, as a result, incorporating OIVs into the state machine might turn out to be a $\mathcal{NP}$-hard problem.

5 Implementation and evaluation

As part of the design cycle of DSR, the mapping has been evaluated on three cases: a) the academic cases EU-Rent [83, 88] and Volley [27], and b) the real-world case Social Housing [66, 89]. At first, this mapping was executed manually, while later a TypeScript reference implementation (automated *converter*) of the mapping[17] was created to allow for easy validation with several DEMO models. Every transformation that was performed with the mapping provided feedback on the correctness and completeness of the mapping. This resulted in several iterations that evolved, improved and enhanced the mapping and its reference implementation so in order to able to deal with all the required concepts as outlined in Sect. 4. The reference implementation eased the process of code generation and evaluation of the mapping; a major redesign of the mapping or its reference implementation was never needed.

Currently the mapping and its reference implementation are being evaluated with two other real-world cases from two other organizations. Up until this moment, no changes had to be made to the mapping nor its reference implementation. As an illustration, in the next sections the input and output for the Social Housing case that was part of the evaluation is shown.

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

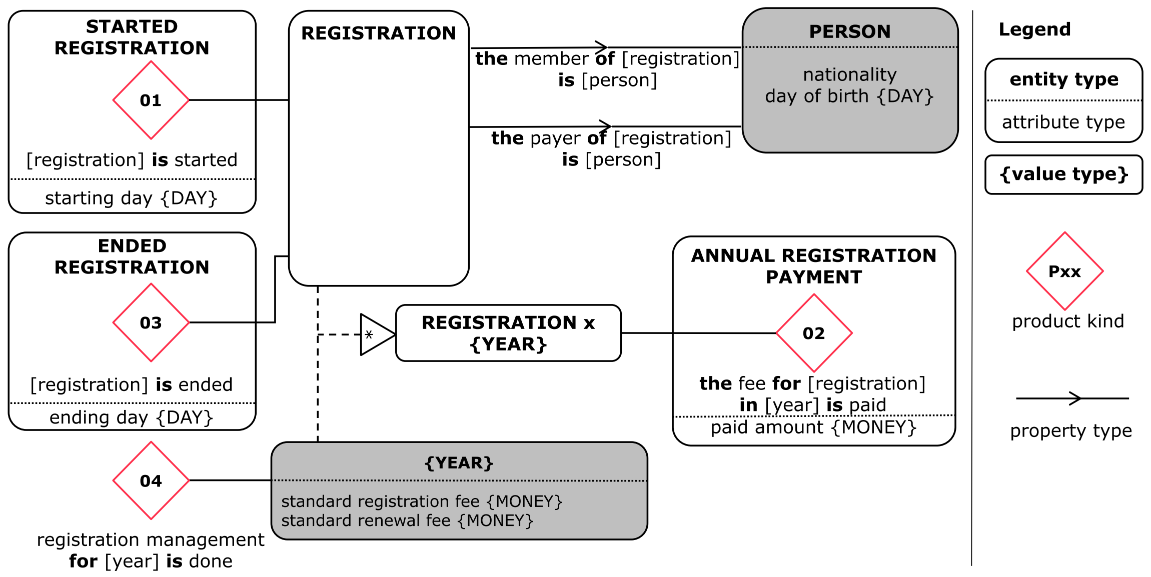

STARTED REGISTRATION
01
[registration] is started
starting day {DAY}
REGISTRATION
the member of [registration] is [person]
the payer of [registration] is [person]
PERSON
nationality
day of birth {DAY}
Legend
entity type
attribute type
{value type}
Pxx
product kind
property type
ENDED REGISTRATION
03
[registration] is ended
ending day {DAY}
REGISTRATION x {YEAR}
ANNUAL REGISTRATION PAYMENT
02
the fee for [registration] in [year] is paid
paid amount {MONEY}
04
registration management for [year] is done
{YEAR}
standard registration fee {MONEY}
standard renewal fee {MONEY}

Fig. 10 OFD for Social Housing

Table 7 DFSs 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 8 ARS01 (executed by TAR01) for Social Housing

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]	[TK01/pm]
			to the performer **of the** request	
	else	decline	registration starting **for** [registration]	[TK01/dc]
			to the performer **of the** request	
			with * *reason for declining* *	

5.1 Social housing

Social Housing is the domain where houses with low(er) rents are provided to people with little or no income. It is of particular interest of ICTU[18], as it's being reorganized every few years and many parties are involved. It is desirable to have an application, for the support of relevant processes within this domain, that can easily be changed to accommodate changes in the responsibilities of the parties involved. 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.

5.1.1 DEMO models

The CSD (Fig. 9) and TPT (Table 6) 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.

The OFD (Fig. 10) 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 7 shows the DFSs to calculate those.

Action rules guide actors in their decisions. Table 8 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 authorized to play their (performer and addressee) role in this request,

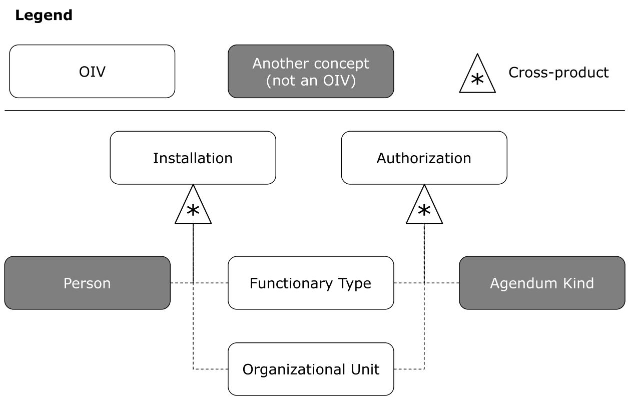


Fig. 11 Chosen OIVs for social housing

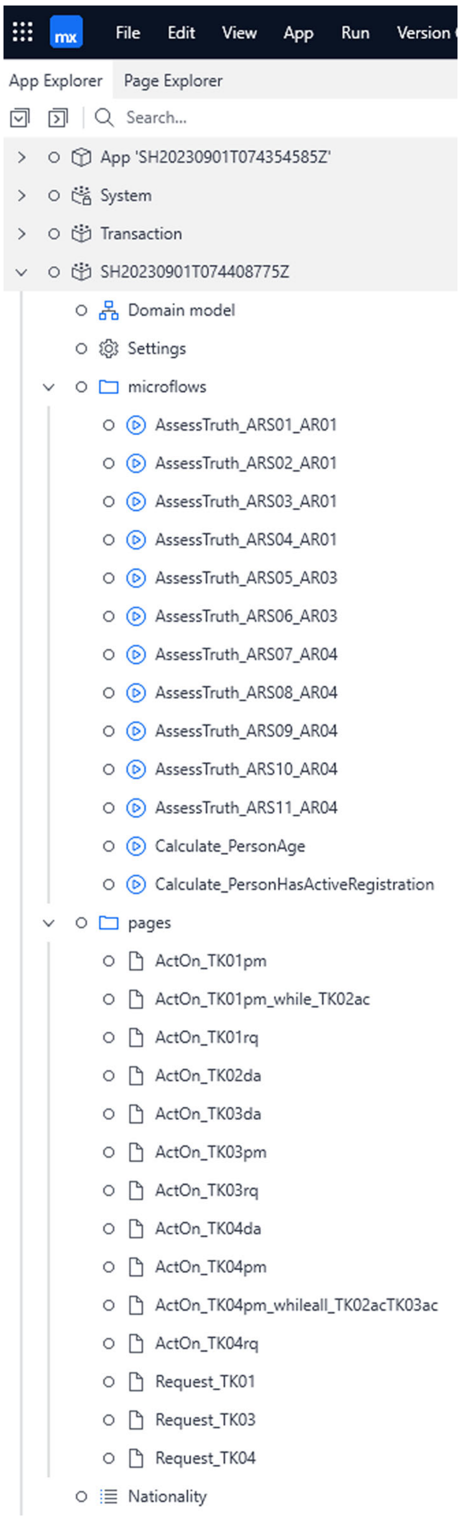
that the (aspirant) member is at least 18 years old and Dutch, that the (aspirant) member doesn't have an active registration at the starting day of the new registration, and that the starting day is in the current year or later—the latter implies one can start a registration retrospectively, but limited 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.

5.1.2 Organization implementation

Having explored the (stable and implementation independent) DEMO model for the registration part of the Social Housing domain, the enterprise implementation and its desired flexibility has been elaborated. As mentioned earlier, within the domain of Social Housing, it is important to be able to easily shift responsibilities between *organizational units* and *functionary types*. In terms of organization units, actor roles can shift from one social housing association to an umbrella of social housing associations, or to a municipality, or may be in the future as a collaboration of municipalities. The *authorization* of a functionary type to fulfill one or more actor role(s) 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 *install* persons to fulfill functionary types (and not actor roles directly), it was decided to choose the following OIVs from the collection of OIVs [67] as the OIVs that should be easily changeable in the supporting software.

– *Organizational Unit*, e.g., 'City of Amsterdam' and 'Woningnet';

[18] Stichting ICT Uitvoeringsorganisatie (ICTU) is 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. See https://www.ictu.nl/.



```
> App 'SH20230901T074354585Z'
> System
> Transaction
v SH20230901T074408775Z
    Domain model
    Settings
  v microflows
      AssessTruth_ARS01_AR01
      AssessTruth_ARS02_AR01
      AssessTruth_ARS03_AR01
      AssessTruth_ARS04_AR01
      AssessTruth_ARS05_AR03
      AssessTruth_ARS06_AR03
      AssessTruth_ARS07_AR04
      AssessTruth_ARS08_AR04
      AssessTruth_ARS09_AR04
      AssessTruth_ARS10_AR04
      AssessTruth_ARS11_AR04
      Calculate_PersonAge
      Calculate_PersonHasActiveRegistration
  v pages
      ActOn_TK01pm
      ActOn_TK01pm_while_TK02ac
      ActOn_TK01rq
      ActOn_TK02da
      ActOn_TK03da
      ActOn_TK03pm
      ActOn_TK03rq
      ActOn_TK04da
      ActOn_TK04pm
      ActOn_TK04pm_whileall_TK02acTK03ac
      ActOn_TK04rq
      Request_TK01
      Request_TK03
      Request_TK04
    Nationality
```

Fig. 12 Project outline of the generated Mendix application for Social Housing

- *Functionary type*, e.g., 'Civil Servant Social Housing' and 'Customer Contact Manager';
- *Authorization*, e.g., functionary type 'Civil Servant Social Housing' in organizational unit 'City of Amsterdam' is authorized to settle agendum kind 'TK01/rq (registration starting is requested)' and 'Customer Contact Manager' in organizational unit 'Woningnet' is authorized to settle agendum kind 'TK01/rq (registration starting is requested)'; and
- *Installation*, e.g., person 'George' is installed as functionary type 'Civil Servant Social Housing' in organizational unit 'City of Amsterdam' and person 'George' is installed as functionary type 'Customer Contact Manager' in organizational unit 'Woningnet'.

This is summarized in Fig. 11 that shows the FM of the implementation. This model can be extended with product kinds (and related transactor roles) to show the process of creating an implementation (design), as suggested in [28].

5.1.3 Generated low-code application

Figures 13 and 14 show the input JSON files for the automated converter that uses these files to generate a Mendix application. Figure 12 shows the project folder of the generated Mendix application, containing pages with buttons, microflows and enumerations. Figure 15 shows the generated domain (data) model consisting of entities, attributes and associations. It also shows the generic (computed but not persisted, therefore orange-colored) entity AssessmentResult that will be used in the microflows (see below). 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 16 shows how the truth part of ARS01 (Table 8) was implemented in a Mendix microflow and Fig. 17 shows a screenshot of the running application where an actor with the right authorization can deal with a C-act of kind TK01/rq.

5.1.4 Achieved flexibility

As the software is easily regenerated from a changed DEMO model, the approach as suggested in this article supports changes in the enterprise's ontology. One could, e.g., easily add the transaction kinds for the assignment of a house, or change the (business) rules of ARS01.

Moreover, by generating a run-time configurable item for each chosen OIV, the resulting application is intrinsically adaptable with regard to implementation design decisions for a given enterprise's ontological model. Specifically for the Social Housing case, it made it possible for the enterprise designer or HR-employee to 'configure' their (implementa-

```
{
    "transactorroles": [
        {"id": "01", "transactionkindname": "registration starting", "actorrolename": "registration starter",
         "focus": true, "casekinds": ["Registration"], "product": "[registration] is started",
         "productname": "StartedRegistration"},
        {"id": "02", "transactionkindname": "registration paying", "actorrolename": "registration payer",
         "focus": false, "casekinds": ["Registration", "Year"],
         "product": "the fee for [registration] in [year] is paid",
         "productname": "AnnualRegistrationPayment"},
        {"id": "03", "transactionkindname": "registration ending", "actorrolename": "registration ender",
         "focus": true, "casekinds": ["Registration"], "product": "[registration] is ended",
         "productname": "EndedRegistration"},
        {"id": "04", "transactionkindname": "registration management",
         "actorrolename": "registration manager", "focus": true, "casekinds": ["Year"],
         "product": "registration management for [year] is done", "productname": "RegistrationManagement"}
    ],
    "compositeactorroles": [
        {"id": "01", "name": "(aspirant) member"}
    ],
    "multipletransactionkinds" : [
        {"id" : "91", "name" : "Costs and terms"},
        {"id" : "92", "name" : "Person & living facts"}
    ],
    "initiates" : [
        {"ctar": "01", "tk": "01"},
        {"ear": "01", "tk": "02"},
        {"ear": "04", "tk": "04"},
        {"ear": "04", "tk": "02"},
        {"ear": "04", "tk": "03"},
        {"ctar": "01", "tk": "03"}
    ],
    "hasaccess" : [
        {"ear": "01", "mtk": "91"},
        {"ear": "01", "mtk": "92"},
        {"ear": "04", "mtk": "01"},
        {"ear": "04", "mtk": "91"}
    ],
    "factkinds": [
        {"name": "Registration", "type": "entitytype", "focus": true},
        {"name": "Person", "type": "entitytype", "focus": false},
        {"name": "Year", "type": "entitytype", "focus": false},
        {"name": "Year", "type": "valuetype"}, {"name": "Day", "type": "valuetype"},
        {"name": "Money", "type": "valuetype"}, {"name": "Number", "type": "valuetype"},
        {"name": "Boolean", "type": "valuetype"}, {"name": "Nationality", "type": "valuetype"},
        {"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": "paid amount", "type": "attributetype", "domain": "AnnualRegistrationPayment",
         "range": "Money"},
        {"name": "day of birth", "type": "attributetype", "domain": "Person", "range": "Day"},
        {"name": "nationality", "type": "attributetype", "domain": "Person", "range": "Nationality"},
        {"name": "year", "type": "attributetype", "domain": "Year", "range": "Year"},
        {"name": "standard registration fee", "type": "attributetype", "domain": "Year", "range": "Money"},
        {"name": "standard renewal fee", "type": "attributetype", "domain": "Year", "range": "Money"},
        {"name": "PersonAge", "type": "derived", "domain": ["Person", "Day"], "range": "Number"},
        {"name": "PersonHasActiveRegistration", "type": "derived", "domain": ["Person", "Day"],
         "range": "Boolean"}
    ],
    "actionrules": [
        {"id": "01", "ear": "01", "when": "01-rq", "while": [], "then": ["01-pm"], "else": ["01-dc"]},
        {"id": "02", "ear": "01", "when": "01-pm", "while": [], "then": ["02-rq"]},
        {"id": "03", "ear": "01", "when": "02-da", "while": [], "then": ["02-ac"], "else": ["02-rj"]},
        {"id": "04", "ear": "01", "when": "01-pm", "while": ["02-ac"], "then": ["01-ex", "01-da"]},
        {"id": "05", "ear": "03", "when": "03-rq", "while": [], "then": ["03-pm"], "else": ["03-dc"]},
        {"id": "06", "ear": "03", "when": "03-pm", "while": [], "then": ["03-ex", "03-da"]},
        {"id": "07", "ear": "04", "when": "04-rq", "while": [], "then": ["04-pm", "04-rq"],
         "else": ["04-dc", "04-rq"]},
        {"id": "08", "ear": "04", "when": "04-pm", "while": [], "then" : ["02-rq, 03-rq"]},
        {"id": "09", "ear": "04", "when": "03-da", "while": [], "then": ["03-ac"], "else": ["03-rj"]},
        {"id": "10", "ear": "04", "when": "04-pm", "whileall": ["02-ac", "03-ac"],
         "then": ["04-ex", "04-da"]},
        {"id": "11", "ear": "04", "when": "04-da", "while": [], "then": ["04-ac"], "else": ["04-rj"]}
    ]
}
```

Fig. 13 DEMO model for social housing in JSON format

Table 9 Limitations of the DEMO to Mendix mapping, structured according to [5] (part 1)

Category	*Type of limitation*	*Discussion*
Input knowledge and technology	L1. Data sources/ Foundations	As reported in Sect. 3, existing knowledge bases on DEMO and low-code platforms are used as a foundation. General limitations follow from the use of DEMO and Mendix, but there are no specific limitations other than the ones reported below
	L2. Insufficient number of previous studies	In Sect. 2 alternative MDSD approaches are discussed. At the same time, not much studies have been done on mapping DEMO to low-code platforms such as Mendix
	L3. Sample size	The artifact has been evaluated on three cases; additionally, two more cases are subject of ongoing evaluations. More cases will be needed to thoroughly evaluate the artifact
	L4. Novelty/ Shortcomings of the technology	Both DEMO as source of the mapping, and Mendix as target of the mapping are well established and future-proof"technologies". No limitations
Resulting process	L5. Setting	The primary goal of the research effort was aimed at prototyping and exploring the (im)possibilities of using semantically rich enterprise models (such as DEMO models) as a base to generate (low-code) applications. See L3
	L6. Participants	Next to the researchers and two developers, domain experts from the real-world case Social Housing were involved
	L7. Method	Given the primary goal of the research (see L5), the evaluations performed are considered (more than) enough. Of course, in moving beyond a prototype stage, further evaluations are needed. Also see Sect. 7.2
	L8. Difficulties—out of the authors' control	As the current evaluations of the mapping, and the Social Housing case in particular, are performed in essentially a "laboratory setting", i.e., away from, e.g., budgetary and political pressures, there were no limitations in this regard. Needless to say, that in moving beyond the prototype stage, such constraints may influence future evaluations in real-world settings

tion) design decisions about "which actor roles are fulfilled by this functionary type" and "which person is fulfilling which functionary type (and at what moment)" directly in the (running) application, without needing to specify it for a developer or translate it to more technical terms such as 'user role' or 'permission rule', and without the need to rebuild or redeploy the application. The approach as suggested in this research thus supports both changes in the enterprise's ontology and its implementation.

6 Limitations

In this section we briefly reflect on the limitations of the research as reported on in this article. As we have followed a DSR approach, we use the typology of DSR limitations as suggested in [5] as a way to structure these reflections. The authors of [5] have based this typology on a structured literature review into "*self-reported*" limitations on design science artifacts. The resulting typology identifies 19 types of limitations which the authors of [5] have grouped into four categories based on [13].

It is also important to note that (self-reported) limitations do not necessarily have an immediate relation to the quality of the designed artifact. Described limitations also clarify and document the limitations one should be aware of when using the artifact in practice, or as a base for further development of a similar artifact; just like the Medication Guides that come with medicines. Using this typology as a base, Tables 9, 10 and 11 summarize the limitations of the created mapping from DEMO to Mendix.

7 Conclusion and further research

In this paper, we investigated the potential benefits of using DEMO ontological enterprise models, supplemented with explicit enterprise implementation design decisions, captured in OIVs, as a semantically rich base for MDSD. As a first step to show this benefit, we focused on the design of a mapping from DEMO models to Mendix for the (automated) creation of a low-code application that also intrinsically accommodates run-time implementation design decisions.

```
{
  "appname": "SH",
  "defaultmodule": "SH",
  "demomodel": "./shdemomodel.json",
  "oivs": [
        { "name": "Functionary Type" },
        { "name": "Organizational Unit" },
        { "name": "Authorization", "concerns": ["Functionary Type", "Organization Unit", "Agendum Kind"] },
        { "name": "Installation", "concerns": ["Functionary Type", "Organization Unit", "Person"] }
  ],
  "valuetypemapping": [
    { "name": "Year", "primitive": "integer" },
    { "name": "Day", "primitive": "datetime" },
    { "name": "Money", "primitive": "decimal" },
    { "name": "Number", "primitive": "integer" },
    { "name": "Boolean", "primitive": "boolean" },
    { "name": "Nationality", "primitive": "string", "values": ["NL","EN"] }
  ]
}
```

Fig. 14 The app descriptor file for the DEMO model for Social Housing in JSON format, containing the name of the application to be generated, the mapping from value types to software primitives, and the OIVs to be taken into consideration

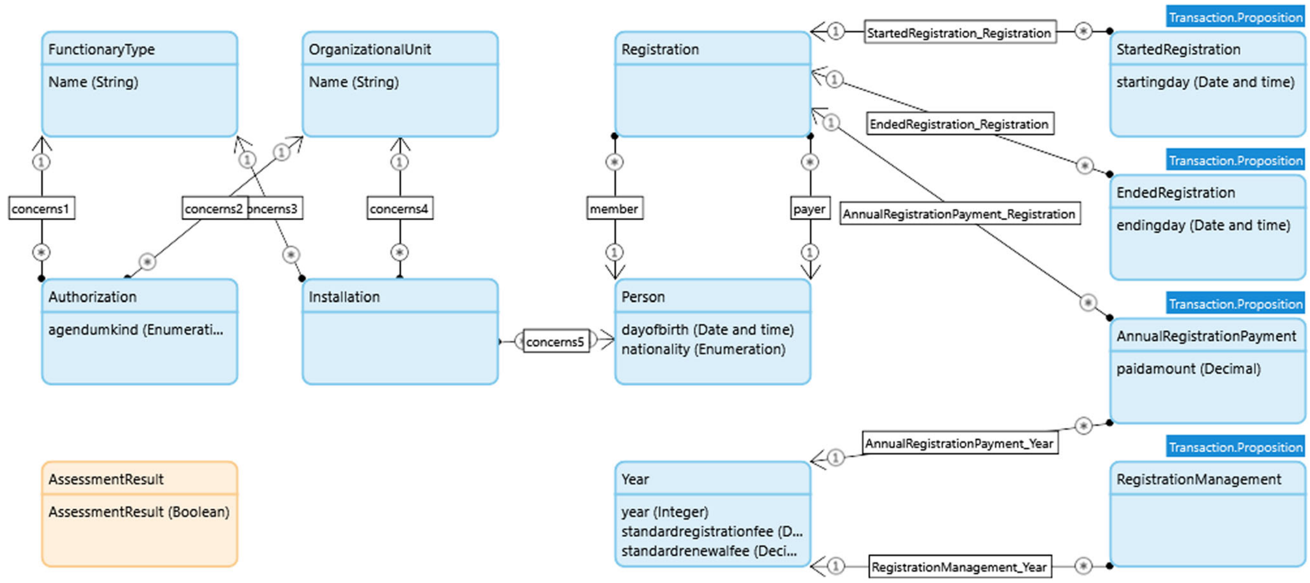
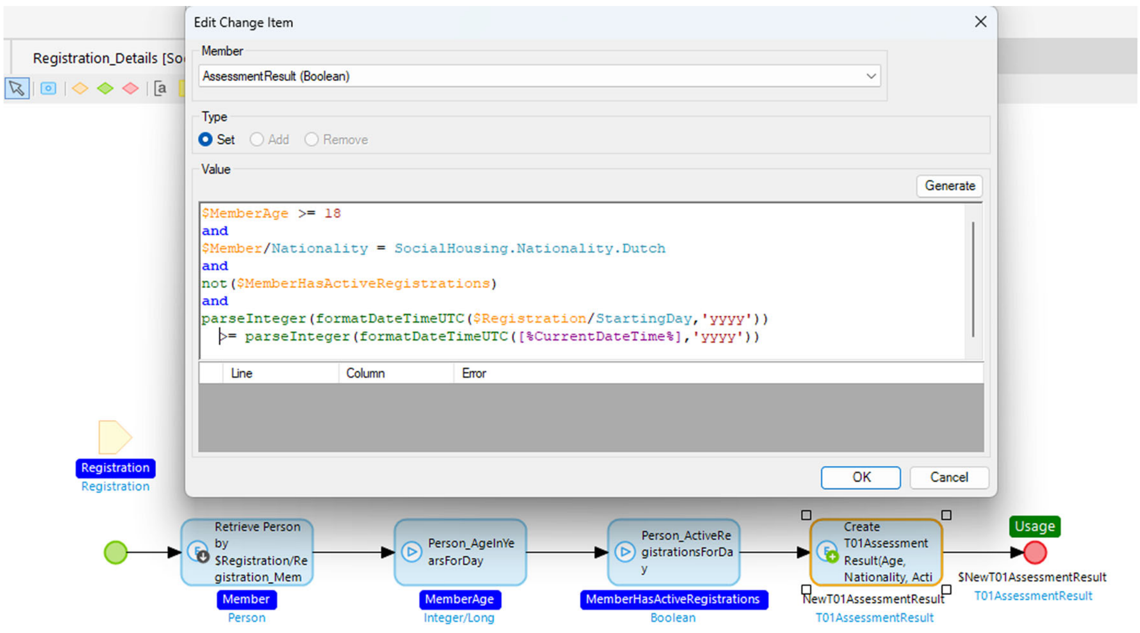

FunctionaryType
Name (String)
OrganizationalUnit
Name (String)
Registration
StartedRegistration_Registration
Transaction.Proposition
StartedRegistration
startingday (Date and time)
EndedRegistration_Registration
Transaction.Proposition
EndedRegistration
endingday (Date and time)
concerns1
concerns2
concerns3
concerns4
member
payer
AnnualRegistrationPayment_Registration
Authorization
agendumkind (Enumerati...
Installation
concerns5
Person
dayofbirth (Date and time)
nationality (Enumeration)
Transaction.Proposition
AnnualRegistrationPayment
paidamount (Decimal)
AnnualRegistrationPayment_Year
Transaction.Proposition
AssessmentResult
AssessmentResult (Boolean)
Year
year (Integer)
standardregistrationfee (D...
standardrenewalfee (Deci...
RegistrationManagement_Year
RegistrationManagement

Fig. 15 Domain (data) model of the generated Mendix application for Social Housing

By introducing the concept of Organization Implementation Variable into the MDSD (code generation) process, the mapping from enterprise implementation design decision to software implementation is made transparent, thereby not leaving it up to developers to make the (right) choice. Moreover, by generating a run-time configurable item for each chosen OIV, the resulting application is intrinsically adaptable at run-time with regard to implementation design decisions for a given enterprise's ontological model. This run-time configurability of implementation decisions has shown to support, or even improve, enterprise agility [89]. Moreover, for new products and services, the resulting (Mendix) application can easily be (re)generated from its changed DEMO model. The approach as suggested in this research thus supports both changes in the enterprise's ontology and in its implementation. In this section the conclusions and future research are further detailed.

7.1 Conclusions

Given the choice for Mendix as low-code target platform, a mapping from the complete DEMO metamodel, including enterprise implementation, to the Mendix metamodel has been designed. This mapping is implemented in TypeScript, using the Mendix SDK, in order to generate a readily deployable low-code application from the DEMO ontological model of an enterprise, including the support for run-time configurable OIVs. In doing so, a reusable component in Mendix is created to support the CTP as well as for showing relevant agenda to the actors that have to deal with them.


Edit Change Item

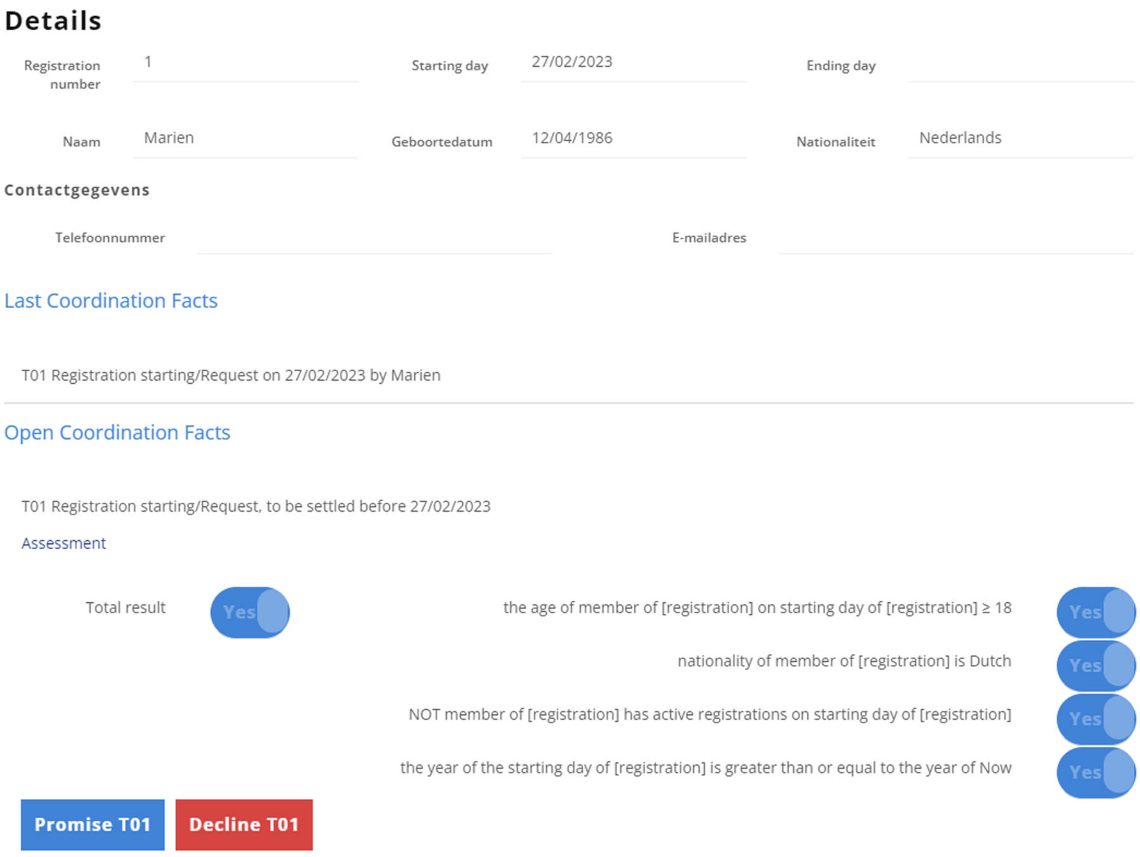
Member: AssessmentResult (Boolean)

Type: [✓ Set] Add Remove

Value:

```
$MemberAge >= 18
and
$Member/Nationality = SocialHousing.Nationality.Dutch
and
not($MemberHasActiveRegistrations)
and
parseInteger(formatDateTimeUTC($Registration/StartingDay,'yyyy'))
  >= parseInteger(formatDateTimeUTC([%CurrentDateTime%],'yyyy'))
```

Fig. 16 Mendix implementation of the ‘truth part’ of ARS01


Details

Registration number	1	Starting day	27/02/2023	Ending day	
Naam	Marien	Geboortedatum	12/04/1986	Nationaliteit	Nederlands

Contactgegevens

Telefoonnummer		E-mailadres	

Last Coordination Facts

T01 Registration starting/Request on 27/02/2023 by Marien

Open Coordination Facts

T01 Registration starting/Request, to be settled before 27/02/2023

Assessment

Total result	Yes
the age of member of [registration] on starting day of [registration] ≥ 18	Yes
nationality of member of [registration] is Dutch	Yes
NOT member of [registration] has active registrations on starting day of [registration]	Yes
the year of the starting day of [registration] is greater than or equal to the year of Now	Yes

Fig. 17 Screen for AR01 to deal with TK01/rq

Table 10 Limitations of the DEMO to Mendix mapping, structured according to [5] (part 2)

Category	*Type of limitation*	*Discussion*
Resulting artifact	L9. Simplifications	The mapping from a DEMO model to a Mendix application as currently reported on was aimed at prototyping and exploring the (im)possibilities of such an approach. This allowed to conduct the evaluation of the mapping in a laboratory setting. See L5 and L8
	L10. Evaluator's bias/Measurement bias	While Social Housing is a real-world case, the generated application is not validated with real end users as the setting was aimed at prototyping and exploring the (im)possibilities of such an approach (see L5). The DEMO model was validated with subject matter experts from the Social Housing case, but not with end users (of the Mendix applications). The resulting application was checked by the creators of the DEMO models to confirm that the application indeed provides a complete and correct implementation that supports the processes, information (items) and business rules as defined in the DEMO model
	L11. Not real users/ Controlled experiment	As discussed in L10, the evaluation was done in an experimental or laboratory setting. This experiment involved two academic cases (EU-Rent and Volley), and one real-world case (Social Housing). Additional real-world case evaluations are ongoing
	L12. Real situation/ Prototype	As discussed in L10, the goals of the research effort as reported on in this article, allowed to conduct the evaluation as a laboratory experiment
	L13. Limited performance	(a) In creating the mapping, the researchers noticed, as reported in [80], that the DEMO metamodel as defined in DEMO-SL [24] is not specified from an operational or software development point of view but rather from a diagramming or modeling perspective. Therefore, in the (manual) conversion to JSON, the parts that are for visualization only are left out. (b) At the same time, there was a need to include other aspects that are relevant for generating software, such as the application name and mapping to software primitives. These aspects, including the choice of OIVs have are separated from the actual DEMO file and put into a descriptor file. (c) The mapping does not include mapping rules for generalization and specialization Entity Types from the DEMO Fact Model. (d) The mapping from Derived Fact Specifications and Action Rule Specifications to a Mendix microflow turned out to not be straightforward. (e) Furthermore, the implementation of OIVs in Mendix, or maybe even software in general, turned out not to be straightforward. The impact of certain choices on the state machine and logic is not completely detailed yet. This detailing is a labor-intensive task that should be done for each OIV independently, but also in different combinations. It can even be that when combining several OIVs, the problem becomes too complex to solve. It is presently not clear whether all OIVs can be implemented completely independent of others, as suggested by Normalized Systems theory [75]. (f) Currently, one Mendix application is created for a given DEMO model. For bigger enterprise models, i.e., larger cases, it might be desired that a portfolio of several separate, but connected, applications is generated
	L14. Requirements to use the artifact	DEMO skills are needed to create the (case specific) DEMO models. Currently, as reported in L13, the DEMO model has to be encoded in JSON manually. Feeding the input file(s) into the automated converter requires a NodeJS environment but no other technical skills. In case one wants to inspect or edit the generated application, Mendix skills are required. Mendix makes the process of deployment relatively easy. In a next iteration, the first two steps should be covered by a proper DEMO modeling tool, while the latter three steps should be available as easy-to-use SaaS

One advantage of the proposed approach follows from the use of the DEMO ontological enterprise model as the starting point. Since these models are claimed to be of high quality, i.e., coherent, comprehensive, consistent and concise, the generated software is of high quality and only contains the necessary constructs to support the enterprise end users.

As (parts of) the software are simply (re)generated when new transaction kinds arise, the proposed approach supports some level of enterprise agility. By adding OIVs to the MDSD approach, it becomes possible to change a predefined set of implementation design decisions at run-time, therefore allowing for even more agility for the enterprise as software development is not a limiting factor anymore in changing the enterprise.

Another advantage of this approach is that the generated application model can be adapted easily through the low-code visual paradigm. This allows for changes in the user interface,, 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 output model can become a source of hidden design decisions.

Table 11 Limitations of the DEMO to Mendix mapping, structured according to [5] (part 3)

Category	*Type of limitation*	*Discussion*
Design knowledge	L15. Uncertainty in future events/ Time- related constraints/ Risks	The current focus of the mapping toward Mendix, is of course tied to the future evolution of the Mendix platform. At the same time, however, the generic insights of mapping from a semantically rich enterprise model in terms of, e.g., DEMO models, to a (low-)code platform such as Mendix remains design knowledge that is transferable to mappings toward other technologies
	L16. DSR outcome not compared to alternatives	It turns out to be quite hard to compare the MDSD approach as applied in this research to ones that use a different kind of input, such as BPMN or UML. There does not seem to be a generic framework to do these kind of comparisons, especially when both input and output are different
	L17. Scope of DSR application	The assumption behind the artifact is that adaptable software is required. In situations that do not need adaptable software—it is hard to imagine in the world we currently live in—this approach might not be valuable. The evaluations suggest the mapping is generically applicable to different kinds of enterprises, see L3
	L18. Other theoretical limitations	Some of the limitations discussed under L13 actually have a theoretical cause. For example, L13.a can be said to be caused by a lack in the formalization of the theory behind DEMO. Also, L13.e suggests missing theoretical insights in general (i.e., not only specific to DEMO) on how different OIVs are related
	L19. Generalizability and transferability	The current mapping is, clearly, specific toward the combination of DEMO as source and the Mendix platform as target. For the goal of the initial mapping (see L9) this is considered defensible. Nevertheless, the obvious first candidate for generalization would be to be able to deal with different target platforms

Reflecting on the MDSD approach as applied in this research, by making explicit the required enterprise implementation flexibility and giving it a specific place in the generated application model, the notions of software adaptability and enterprise agility have been connected, thereby at least improving the latter. As the application can be regenerated easily, changing the software is not the limiting factor anymore in changing the enterprises service and product portfolio or its implementation.

7.2 Future research

As already hinted at in the discussion of the limitations (Sect. 6), further work is called for. With regards to the limitations of type L13 (Limited performance), we, respectively, foresee:

a) There is a need to better detail the generic DEMO metamodel, containing only the true DEMO concepts and separating the core of DEMO from the purpose-specific information such as visualization or code generation.
b) To arrive at (low-)code that remains readable and understandable by humans, it is important to generate meaningful application names and other mappings to relevant software primitives. More work is needed to understand what exactly is needed on top of the DEMO model to generate a working and meaningful application.
c) The mapping does not include mapping rules for generalization and specialization Entity Types from the DEMO Fact Model. More cases are needed to be able to define such a mapping.
d) The mapping from Derived Fact Specifications and Action Rule Specifications to a Mendix microflow is not straightforward. It is suggested to further detail that mapping for all possible DEMO constructs. Possibly, it turns out to be easier to manually perform such a mapping case-by-case than to generalize it to an automatable mapping.
e) More work is needed regarding the implementation of OIVs in Mendix, and software in general. It is not clear whether all OIVs can be implemented completely independent of others, as suggested by Normalized Systems theory [75]. More case studies are needed, involving more OIVs, to fully understand the complexity of this problem.
f) Currently, one Mendix application is created for a given DEMO model. For bigger enterprise models, i.e., larger cases, it might be desired that a portfolio of several separate, but connected, applications is generated. One solution is of course to split up the DEMO models and generate an application for each separate model (part). Another solution is to create a sort of microservice landscape with small applications, perhaps on the level of a

single transaction kind, as suggested by several authors [66, 86, 116]. It is worthwhile to find out what level of granularity would be optimal and how to introduce the splitting of an enterprise model into different applications.

As stated for limitation type L16, it is hard to compare the MDSD approach as applied in this research to ones that use a different kind of input, such as BPMN or UML. There does not seem to be a generic framework to do these kind of comparisons, especially when both input and output is different. It could, therefore, be interesting to research ways to compare different approaches toward MDSD.

The current mapping is clearly specifically targeted toward the Mendix platform (see L19). 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. This conforms to the need to (better) distinguish between the PIM and PSM as defined by MDA. Further research is needed to get a perspective on the feasibility and usability of such an abstraction.

Next to that, new features of the Mendix platform also provide opportunities for future improvements of the mapping:

a) Mendix has recently launched native workflow capability in the platform. As the majority of the current mapping and its implementation was set up before that time, it does not use this feature. It could be interesting to look into these possibilities and see how it can support the CTP. As this component is created as a generic component, it is easy to rebuild without impacting the model converter.
b) Mendix has recently also introduced ‘Mendix Connect’, that eases integration with (existing) external (data) sources through APIs. As it does not seem to be available in the SDK, it is not incorporated in the TypeScript implementation of the mapping. It could be interesting to see how this feature can be used to accommodate integration of external data.

Finally, the use of Large Language Models (LLMs) and (generative) Artificial Intelligence (AI) in software development is growing fast [99]. And while there are several potential advantages of such an approach, including increased efficiency and accuracy,[19] it is unlikely that it will fully replace software developers in the near future, thus still allowing for implicit hard-coded implementation decisions. Moreover, these techniques are black-box, i.e., it is unknown how exactly input is transformed into software. This is a great issue in situations where traceability is needed, e.g., when executing laws, something that is easily achieved with MDSD in general, and the MDSD-based approach as applied in this research. Use cases where LLMs and AI could be able to add value to the outlined approach:

a) Enhance modeling tools so that similar advantages as for software development can be achieved, but on the level of enterprise modeling;
b) Transform text (prompts) into an enterprise model to allow users to create an enterprise model without having to learn the specific modeling language;
c) Support the making of implementation optimization decisions.

These use cases are on the level of creating the input model (CIM; in our case: DEMO) and it would be interesting to see how such approaches can be integrated into the approach as outlined in this research to create a wider-scoped approach for enterprise (model-driven) software development.

[19] https://www.linkedin.com/pulse/chatgpt-how-revolutionize-software-engineering-industry-lopez/.

Acknowledgements We would like to thank the anonymous reviewers from both the Software and Systems Modeling journal and the Practice of Enterprise Modeling (PoEM 2022) conference. Their feedback has resulted in many improvements to the original article. We would like to thank the participants of PoEM 2022 for their participation in the discussions regarding the original version of this article. These discussions have also provided us with additional inspirations for improvements to this article.

Funding Open access funding provided by TU Wien (TUW).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. **45**(3), 515–526 (2006). https://doi.org/10.1147/sj.453.0515
2. Aßmann, U., Zschaler, S., Wagner, G.: Ontologies, Meta-models, and the Model-Driven Paradigm. In: Calero, C., Ruiz, F., Piattini, M. (eds.) Ontologies for Software Engineering and Software Technology, pp. 249–273. Springer, Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/3-540-34518-3_9
3. Austin, J.L.: How to do things with words. Oxford University Press, William James Lectures (1962)
4. Aveiro, D., Pinto, D.: A case study based new DEMO way of working and collaborative tooling. In: 2013 IEEE 15th Confer-

ence on Business Informatics, pp. 21–26 (2013). https://doi.org/10.1109/CBI.2013.12
5. Barata, J., da Cunha, P.R., de Figueiredo, A.D.: Self-reporting limitations in information systems design science research. Bus. Inf. Syst. Eng. **65**, 143–160 (2023). https://doi.org/10.1007/s12599-022-00782-8
6. Basha, N.M.J., Moiz, S.A., Rizwanullah, M.: Model based software development: issues challenges. Int. J. Comput. Sci. Inform. (2013). https://doi.org/10.47893/IJCSI.2013.1123
7. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model Transformations? Transformation Models! In: O. Nierstrasz, J. Whittle, D. Harel, G. Reggio (eds.) Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1–6, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 4199, pp. 440–453. Springer (2006). https://doi.org/10.1007/11880240_31
8. Biehl, M.: Literature Study on Model Transformations. Tech. rep, Royal Institute of Technology Stockholm, Sweden (2010)
9. Bock, A.C., Frank, U.: 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), pp. 57–66 (2021). https://doi.org/10.1109/MODELS-C53483.2021.00016
10. van Bockhooven, S., Op 't Land, M.: 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, *CEUR Workshop Proceedings*, vol. 1408. Lisbon, Portugal (2015). http://ceur-ws.org/Vol-1408/paper3-tee.pdf
11. Borst, W.: Construction of engineering ontologies. Ph.D. thesis, Institute forTelematica and Information Technology, University of Twente (1997)
12. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice, second edn. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers (2017). https://doi.org/10.2200/S00441ED1V01Y201208SWE001
13. Brocke, J.V., Winter, R., Hevner, A., Maedche, A.: Accumulation and Evolution of Design Knowledge in Design Science Research: A Journey Through Time and Space. Journal of the Association for Information Systems **21**(3), 520–544 (2020). https://doi.org/10.17705/1jais.00611
14. Broy, M., Cengarle, M.V.: UML formal semantics: lessons learned. Softw. Syst. Model. **10**, 441–116 (2011). https://doi.org/10.1007/s10270-011-0207-y
15. Cabot, J.: 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. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3417990.3420210
16. Caetano, A., Assis, A., Tribolet, J.: Using DEMO to analyse the consistency of business process models. In: Advances in Enterprise Information Systems II, pp. 133–146. CRC Press, Boca Raton, FL (2012). https://doi.org/10.1201/b12295-17
17. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–645 (2006). https://doi.org/10.1147/sj.453.0621
18. D'aveni, R.A.: Hypercompetition. Simon & Schuster (2010)
19. De Vries, M.: DEMO and the Story-Card Method: Requirements Elicitation for Agile Software Development at Scale. In: R. Buchmann, D. Karagiannis, M. Kirikova (eds.) PoEM 2018: The Practice of Enterprise Modeling, *Lecture Notes in Business Information Processing*, vol. 335, pp. 138–153. Springer (2018). https://doi.org/10.1007/978-3-030-02302-7_9
20. Décosse, C., Molnar, W.A., Proper, H.A.: 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, *Lecture Notes in Business Information Processing*, vol. 174, pp. 16–30 (2014). https://doi.org/10.1007/978-3-319-06505-2_2
21. Di Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., Wimmer, M.: Low-code development and model-driven engineering: Two sides of the same coin? Softw. Syst. Model. **21**, 437–446 (2022). https://doi.org/10.1007/s10270-021-00970-2
22. Dietz, J.L.G.: Enterprise Ontology-Theory and methodology. Springer, Berlin (2006) https://doi.org/10.1007/3-540-33149-2
23. Dietz, J.L.G.: Architecture - Building strategy into design. Netherlands Architecture Forum, Academic Service - SDU, The Hague, The Netherlands (2008)
24. Dietz, J.L.G.: The DEMO Specification Language v4.7. Tech. rep., Enterprise Engineering Institute (2022). https://ee-institute.org/download/demo-specification-language-4-7-1/
25. Dietz, J.L.G., Hoogervorst, J.A.P.: Enterprise ontology and enterprise architecture-how to let them evolve into effective complementary notions. GEAO J. Enterp. Architect. **2007**, 1 (2007)
26. Dietz, J.L.G., Hoogervorst, J.A.P., Albani, A., Aveiro, D., Babkin, E., Barjis, J., Caetano, A., Huysmans, P., Iijima, J., van Kervel, S., Mulder, H., Op 't Land, M., Proper, H.A., Sanz, J., Terlouw, L., Tribolet, J., Verelst, J., Winter, R.: The discipline of enterprise engineering. Int. J. Organ. Des. Eng. **3**(1), 86–114 (2013). https://doi.org/10.1504/IJODE.2013.053669
27. Dietz, J.L.G., Mulder, J.B.F.: Enterprise ontology—a human-centric approach to understanding the essence of organisation. The Enterprise Engineering Series. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38854-6
28. Dietz, J.L.G., Op 't Land, M., Krouwel, M.R., Mulder, J.B.F.: Enterprise Design. Springer (2024). Forthcoming
29. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008). https://doi.org/10.1016/j.infsof.2008.02.006
30. Durdik, Z.: Architectural Design Decision Documentation through Reuse of Design Patterns. Ph.D. thesis, Institut für Programmstrukturen und Datenorganisation (IPD) (2014)
31. Ettema, R., Dietz, J.L.G.: ArchiMate and DEMO - Mates to Date? In: A. Albani, J. Barjis, J.L.G. Dietz (eds.) Advances in Enterprise Engineering III. CIAO! EOMAS 2009, *Lecture Notes in Business Information Processing*, vol. 34. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01915-9_13
32. Falbo, R.A., Guizzardi, G., Duarte, K., Natali, A.: Developing software for and with reuse: an ontological approach. In: ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications, pp. 311–316. International Association for Computer and Information Science (ACIS) (2002)
33. Fill, H.G., Karagiannis, D.: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. Enterp. Modell. Inf. Syst. Archit. **8**(1), 4–24 (2013)
34. Fox, M.S., Barbuceanu, M., Gruninger, M., Lin, J.: An organization ontology for enterprise modelling. In: Prietula, M., Carley, K., Gasser, L. (eds.) Simulating Organizations: Computational Models of Institutions and Groups, pp. 131–152. MIT Press, Cambridge, MA (1997)
35. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: Future of Software Engineering (FOSE '07), pp. 37–54 (2007). https://doi.org/10.1109/FOSE.2007.14
36. France, R.B., Ghosh, S., Dinh-Trong, T., Solberg, A.: Model-driven development using UML 2.0: promises and pitfalls. Computer **39**(2), 59–66 (2006). https://doi.org/10.1109/MC.2006.65
37. Frank, U.: The MEMO Meta-Metamodel. Universität Koblenz-Landau - Institut für Wirtschaftsinformatik, Tech. rep. (1998)

38. Frank, U., Maier, P., Bock, A.: Low code platforms: promises, concepts andprospects. a comparative study of ten systems. Tech. Rep. 70, ICB-Research (2021). https://doi.org/10.17185/duepublico/75244
39. Geskus, J., Dietz, J.L.G.: developing quality management systems with DEMO. In: A. Albani, J. Barjis, J.L.G. Dietz (eds.) Advances in Enterprise Engineering III. CIAO! EOMAS 2009, *Lecture Notes in Business Information Processing*, vol. 34, pp. 130–142. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01915-9_10
40. Giachetti, R.E.: Design of enterprise systems: theory, architecture, and methods, 1st edn. CRC Press, Boca Raton, FL (2010)
41. Gouveia, D., Aveiro, D.: Two protocols for DEMO engines: PSI or Tell&Agree. In: 15th CIAO! Doctoral Consortium Workshop (2015)
42. Gray, T., Bork, D., De Vries, M.: A new DEMO modelling tool that facilitates model transformations. In: Enterprise, Business-Process and Information Systems Modeling, pp. 359–374. Springer, Heidelberg, Germany (2020). https://doi.org/10.1007/978-3-030-49418-6_25
43. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993). https://doi.org/10.1006/knac.1993.1008
44. Guarino, N., Oberle, D., Staab, S.: What is an ontology? In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, International Handbooks on Information Systems, pp. 1–17. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3_0
45. Guizzardi, G.: Ontological foundations for structural conceptual models. Ph.D. thesis, University of Twente (2005)
46. Habermas, J.: The theory of communicative action. Polity Press, Cambridge (1986)
47. Happel, H.J., Seedorf, S.: Applications of ontologies in software engineering. In: Proceedings of Workshop on Sematic Web Enabled Software Engineering (SWESE), pp. 5–9 (2006)
48. Heitmeyer, C., Shukla, S., Archer, M., Leonard, E.: On model-based software development. In: Münch, J., Schmid, K. (eds.) Perspectives on the Future of Software Engineering, pp. 49–60. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-37395-4_4
49. Hendriks, D.: The selection process of model based platforms. Master's thesis, Radboud Universiteit Nijmegen (2017)
50. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
51. Hinkelmann, K., Gerber, A., Karagiannis, D., Thoenssen, B., van der Merwe, A., Woitsch, R.: A new paradigm for the continuous alignment of business and IT: combining enterprise architecture modelling and enterprise ontology. Comput. Ind. **79**, 77–86 (2016). https://doi.org/10.1016/j.compind.2015.07.009
52. van der Horst, P.: From business transactions to business processes workflows: Using DEMO and BPMN. Master's thesis, Delft University of Technology (2010)
53. Huysmans, P.: On the feasibility of normalized enterprises: applying normalized systems theory to the high-level design of enterprises. Ph.D. thesis, University of Antwerp (2011)
54. Huysmans, P., Ven, K., Verelst, J.: Using the DEMO methodology for modeling open source software development processes. Inf. Softw. Technol. **52**(6), 656–671 (2010). https://doi.org/10.1016/j.infsof.2010.02.002
55. de Jong, J.: A method for enterprise ontology based design of for enterprise information systems. Ph.D. thesis, TU Delft (2013)
56. de Jong, J., Dietz, J.L.G.: Understanding the realization of organizations. In: Aalst, W., Mylopoulos, J., Sadeh, N.M., Shaw, M.J., Szyperski, C., Albani, A., Dietz, J.L.G. (eds.) Advances in Enterprise Engineering IV. Lecture Notes in Business Information Processing, vol. 49, pp. 31–49. Springer, Berlin Heidelberg (2010)
57. Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varró, D.: Survey and classification of model transformation tools. Softw. Syst. Model. **18**, 2361–2397 (2019). https://doi.org/10.1007/s10270-018-0665-6
58. Kamkankaew, P., Phattarowas, V., Khumwongpin, S., Limpiaongkhanan, P., Sribenjachot, S.: Increasing competitive environment dynamics and the need of hyper-competition for businesses. Int. J. Sociol. Anthropol. Sci. Rev. **2**(5), 9–20 (2022). https://doi.org/10.14456/jsasr.2022.36
59. Kardos, M., Drozdova, M.: Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA). J. Inf. Organ. Sci. **34**(1), 89–99 (2010)
60. van Kervel, S.J.H.: Ontology driven Enterprise Information Systems Engineering. Ph.D. thesis, TU Delft (2012)
61. van Kervel, S.J.H., Dietz, J.L.G., Hintzen, J., van Meeuwen, T., Zijlstra, B.: Enterprise ontology driven software engineering. In: Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT-2012), pp. 205–210 (2012). https://doi.org/10.5220/0004080902050210
62. de Kinderen, S., Gaaloul, K., Proper, H.A.: On transforming DEMO models to archimate. In: I. Bider, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, S. Wrycza (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 270–284. Springer Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-31072-0_19
63. Kleidermacher, D., Kleidermacher, M.: Embedded systems security - practical methods for safe and secure software and systems development. Newnes (2012)
64. Kossak, F., Illibauer, C., Geist, V., Kubovy, J., Natschläger, C., Ziebermayr, T., Kopetzky, T., Freudenthaler, B., Schewe, K.D.: A rigorous semantics for BPMN 2.0 process diagrams, chap. A Rigorous Semantics for BPMN 2.0 Process Diagrams, pp. 29–152. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-09931-6_4
65. Krouwel, M.R.: On the design of enterprise ontology-driven software development. Ph.D. thesis, Maastricht University (2023). Approved for publication
66. Krouwel, M.R., Op 't Land, M.: Business driven micro service design - an enterprise ontology based approach to API specifications. In: D. Aveiro, H.A. Proper, S. Guerreiro, M. De Vries (eds.) Advances in Enterprise Engineering XV, *Lecture Notes in Business Information Processing*, vol. 441, pp. 95–113. Springer (2021). https://doi.org/10.1007/978-3-031-11520-2_7
67. Krouwel, M.R., Op 't Land, M., Offerman, T.: Formalizing organization implementation. In: D. Aveiro, R. Pergl, D. Gouveia (eds.) EEWC 2016: Advances in Enterprise Engineering X, *Lecture Notes in Business Information Processing*, vol. 252, pp. 3–18. Springer, Funchal, Madeira Island, Portugal (2016). https://doi.org/10.1007/978-3-319-39567-8_1
68. de Laat, L., Op 't Land, M., Krouwel, M.R.: Supporting goal-oriented organizational implementation - combining DEMO and process simulation in a practice-tested method. In: D. Aveiro, R. Pergl, D. Gouveia (eds.) EEWC 2016: Advances in Enterprise Engineering X, *Lecture Notes in Business Information Processing*, vol. 252, pp. 19–33. Springer (2016). https://doi.org/10.1007/978-3-319-39567-8_2
69. Lankhorst, M., et al.: Enterprise architecture at work modelling communication and analysis. In: The Enterprise Engineering Series, 4th edn. Springer, Berlin (2017). https://doi.org/10.1007/978-3-662-53933-0
70. Lantow, B., Sandkuhl, K., Stirna, J.: Enterprise modeling with 4EM: perspectives and method. In: Karagiannis, D., Lee, M., Hinkelmann, K., Utz, W. (eds.) Domain-Specific Conceptual

Modeling, pp. 95–120. Springer, Berlin (2022). https://doi.org/10.1007/978-3-030-93547-4_5
71. Liddle, S.W.: Model-driven software development. In: D.W. Embley, B. Thalheim (eds.) Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges, chap. Model-Driven Software Development, pp. 17–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-15865-0_2
72. Lindskov, A.: Hypercompetition: a review and agenda for future research. Compet. Rev. **32**(3), 391–427 (2022). https://doi.org/10.1108/CR-06-2021-0097
73. Luitwieler, R.: a selection method for model-driven development tools. Master's thesis, TU Delft (2010)
74. Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J.: Characteristics and challenges of low-code development: the practitioners' perspective. CoRR (2021). https://doi.org/10.48550/arXiv.2107.07482
75. Mannaert, H., Verelst, J., De Bruyn, P.: Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design. Normalized Systems Institute, Koppa (2016)
76. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decis. Support Syst. **15**(4), 251–266 (1995). https://doi.org/10.1016/0167-9236(94)00041-2
77. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electron. Notes Theor. Comput. Sci. (2006). https://doi.org/10.1016/j.entcs.2005.10.021
78. Mráz, O., Náplava, P., Pergl, R., Skotnica, M.: Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method. In: D. Aveiro, R. Pergl, G. Guizzardi, J. Almeida, R. Magalhães, H. Lekkerkerk (eds.) Advances in Enterprise Engineering XI: EEWC 2017, *Lecture Notes in Business Information Processing*, vol. 284, pp. 85–98. Springer Cham (2017). https://doi.org/10.1007/978-3-319-57955-9_7
79. Mulder, J.B.F.: Rapid enterprise design. Ph.D. thesis, Delft University of Technology (2006)
80. Mulder, M.A.T.: Enabling the automatic verification and exchange of DEMO models. Ph.D. thesis, Radboud University Nijmegen (2022). https://repository.ubn.ru.nl/handle/2066/247698
81. Noureen, A., Amjad, A., Azam, F.: Model driven architecture-issues, challenges and future directions. J. Softw. **11**(9), 924–933 (2016)
82. Object Management Group: Model Driven Architecture (MDA) Guide. Tech. rep., Object Management Group (2003). https://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf. Version 1.0
83. Object Management Group: Business Motivation Model. Tech. rep., Object Management Group (2008). http://www.omg.org/spec/BMM/1.0/PDF/. Version 1.0
84. Object Management Group: Model Driven Architecture (MDA) Guide. Tech. rep., Object Management Group (2014). https://www.omg.org/cgi-bin/doc?ormsc/14-06-01. Rev. 2.0
85. van Oosterhout, M.P.A.: Business agility and information technology in service organizations. Ph.D. thesis, Erasmus University Rotterdam (2010)
86. Op 't Land, M.: Applying architecture and ontology to the splitting and allying of enterprises. Ph.D. thesis, Delft University of Technology (2008). http://resolver.tudelft.nl/uuid:0edd0472-39df-4296-b692-e9916e79fb1e
87. Op 't Land, M., Dietz, J.L.G.: Benefits of enterprise ontology in governing complex enterprise transformations. In: A. Albani, D. Aveiro, J. Barjis (eds.) Proceedings of the 2^{nd} Enterprise Engineering Working Conference (EEWC-2012), LNBIP 110, pp. 77–92. Springer, Heidelberg (2012). http://resolver.tudelft.nl/uuid:18d90cf6-fddc-48ae-8101-e9eda186a72c
88. Op 't Land, M., Krouwel, M.R.: Exploring organizational implementation fundamentals. In: H.A. Proper, D. Aveiro, K. Gaaloul (eds.) EEWC 2013: Advances in Enterprise Engineering VII, *Lecture Notes in Business Information Processing*, vol. 146, pp. 28–42. Springer-Verlag Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-38117-1_3
89. Op 't Land, M., Krouwel, M.R., Gort, S.: 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: D. Aveiro, G. Guizzardi, R. Pergl, H.A. Proper (eds.) EEWC 2020: Advances in Enterprise Engineering XIV, *Lecture Notes in Business Information Processing*, vol. 411, pp. 228–242. Springer (2021). https://doi.org/10.1007/978-3-030-74196-9_13
90. Ouyang, C., Dumas Menjivar, M., Van Der Aalst, W., ter Hofstede, A., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. **19**(1), 21–37 (2009)
91. Overby, E., Bharadwaj, A., Sambamurthy, V.: Enterprise agility and the enabling role of information technology. Eur. J. Inf. Syst. **15**(2), 120–131 (2006). https://doi.org/10.1057/palgrave.ejis.3000600
92. Parviainen, P., Takalo, J., Teppola, S., Tihinen, M.: Model-driven development: processes and practices. No. 114 in VTT Working Papers. VTT Technical Research Centre of Finland (2009)
93. Páscoa, C., Sousa, P., Tribolet, J.: Ontology construction: representing dietz "Process" and "State" models using BPMN diagrams. In: M.M. Cruz-Cunha, J. Varajao (eds.) Enterprise Information Systems Design, Implementation and Management: Organizational Applications. Information Science Reference (2010). https://doi.org/10.4018/978-1-61692-020-3.ch004
94. Proper, H.A., Op 't Land, M.: 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, pp. 193–216. Springer (2010)
95. van Reijswoud, V.E., Mulder, J.B.F., Dietz, J.L.G.: Communicative action based business process and information modelling with DEMO. Inf. Syst. J. **9**(2), 117–138 (1999)
96. Richardson, C., Rymer, J.: New development platforms emerge for customer-facing applications. Tech. rep, Forrester (2014)
97. Richardson, C., Rymer, J.R.: Vendor landscape: the fractured. Fertile Terrain Of Low-Code Application Platforms. Tech. rep, Forrester (2016)
98. Rokis, K., Kirikova, M.: Challenges of low-code/no-code software development: a literature review. In: E. Nazaruka, K. Sandkuhl, U. Seigerroth (eds.) Perspectives in Business Informatics Research, *Lecture Notes in Business Information Processing*, vol. 462, pp. 3–17. Springer International Publishing (2022). https://doi.org/10.1007/978-3-031-16947-2_1
99. Ross, S.I., Martinez, F., Houde, S., Muller, M., Weisz, J.D.: 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, pp. 491–514. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3581641.3584037
100. Sanchis, R., García-Perales, Ó., Fraile, F., Poler, P.: Low-code as enabler of digital transformation in manufacturing industry. Appl. Sci. **10**, 12 (2019). https://doi.org/10.3390/app10010012
101. Sandkuhl, K., Stirna, J., Persson, A., Wißotzki, M.: Enterprise modeling: tackling business challenges with the 4EM method. Springer (2014). https://doi.org/10.1007/978-3-662-43725-4
102. Santiago, I., Jiménez, Á., Vara, J.M., de Castro, V., Bollati, V.A., Marcos, E.: Model-driven engineering as a new landscape for traceability management: a systematic literature review. Inf.

Softw. Technol. **54**(12), 1340–1356 (2012). https://doi.org/10.1016/j.infsof.2012.07.008
103. Scheer, A.W., Nüttgens, M.: ARIS architecture and reference models for business process management. In: W.M.P. van der Aalst, J. Desel, A. Oberweis (eds.) Business Process Management, Models, Techniques, and Empirical Studies (BPM2000), *Lecture Notes in Computer Science*, vol. 1806, pp. 376–389. Springer (2000). https://doi.org/10.1007/3-540-45594-9_24
104. Searle, J.R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge, London (1969)
105. Sebastián, G., Gallud, J.A., Tesoriero, R.: Code generation using model driven architecture: a systematic mapping study. J. Comput. Lang. (2020). https://doi.org/10.1016/j.cola.2019.100935
106. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Softw. **20**(5), 42–45 (2003). https://doi.org/10.1109/MS.2003.1231150
107. Shutov, A., Laryushina, Y., Ponomarev, N., Radzinskaia, O.: Extending DEMO with an opportunity for simulation. In: S. Wrycza (ed.) Information Systems Development and Applications. University of Gdańsk (2015)
108. Sijtstra, J.: Quantifying low-code development platforms effectiveness in the Dutch public sector. mathesis, Leiden University (2022). https://theses.liacs.nl/2221
109. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge, MA, USA (1996)
110. Singh, Y., Sood, M.: Model driven architecture: a perspective. In: 2009 IEEE International Advance Computing Conference, pp. 1644–1652 (2009). https://doi.org/10.1109/IADCC.2009.4809264
111. Skotnica, M., van Kervel, S.J.H., Pergl, R.: Towards the ontological foundations for the software executable DEMO action and fact models. In: D. Aveiro, R. Pergl, D. Gouveia (eds.) Advances in Enterprise Engineering X (EEWC 2016), *Lecture Notes in Business Information Processing*, vol. 252, pp. 151–165. Springer (2016). https://doi.org/10.1007/978-3-319-39567-810
112. Spyns, P., Meersman, R., Jarrar, M.: Data modelling versus ontology engineering. SIGMOD Rec. **31**(4), 12–17 (2002). https://doi.org/10.1145/637411.637413
113. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering. Management. John Wiley & Sons Inc, Hoboken, NJ, United States (2006)
114. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: principles and methods. Data Knowl. Eng. **25**(1), 161–197 (1998). https://doi.org/10.1016/S0169-023X(97)00056-6
115. Sushil, Stohr, E.A. (eds.): The flexible enterprise. Flexible Systems Management. Springer India (2014). https://doi.org/10.1007/978-81-322-1560-8
116. Terlouw, L.I., Albani, A.: An enterprise ontology-based approach to service specification. IEEE Trans. Serv. Comput. **6**(1), 89–101 (2013). https://doi.org/10.1109/TSC.2011.38
117. The Open Group: ArchiMate 3.2 Specification. Van Haren (2023)
118. Thomke, S.H.: The role of flexibility in the development of new products: an empirical study. Res. Policy **26**(1), 105–119 (1997). https://doi.org/10.1016/S0048-7333(96)00918-3
119. Uschold, M., King, M.: 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)
120. Van Nuffel, D., Mulder, H., van Kervel, S.: Enhancing the formal foundations of BPMN by enterprise ontology. In: A. Albani, J. Barjis, J.L.G. Dietz (eds.) CIAO! 2009, EOMAS 2009: Advances in Enterprise Engineering III, *Business Information Processing*, vol. 34, pp. 115–129. Springer (2009). https://doi.org/10.1007/978-3-642-01915-9_9
121. Waszkowski, R.: Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine **52**(10), 376–381 (2019). https://doi.org/10.1016/j.ifacol.2019.10.060. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019
122. Weigand, H.: Two decades of language/action perspective. Nat. Lang. Eng. **49**, 45–46 (2006)
123. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43839-8
124. Wu, F., Priscilla, L., Gao, M., Caron, F., De Roover, W., Vanthienen, J.: Modeling decision structures and dependencies. In: On the Move to Meaningful Internet Systems: OTM 2012 Workshops., *Lecture Notes in Computer Science*, vol. 7567 (2012). https://doi.org/10.1007/978-3-642-33618-8_69
125. Yourdon, E.: Just enough structured analysis. Yourdon (2006)
126. Yu, X., Zhang, Y., Zhang, T., Wang, L., Zhao, J., Zheng, G., Li, X.: Towards a model driven approach to automatic BPEL generation. In: D.H. Akehurst, R. Vogel, R.F. Paige (eds.) ECMDA-FA 2007: Model Driven Architecture - Foundations and Applications, *Lecture Notes in Computer Science*, vol. 4530, pp. 204–218 (2007). https://doi.org/10.1007/978-3-540-72901-3_16
127. Zhu, K.X., Zhou, Z.Z.: Research note - lock-in strategy in software competition: open-source software vs. proprietary software. Inf. Syst. Res. **23**(2), 536–545 (2011). https://doi.org/10.1287/isre.1110.0358
128. Zrnec, A.Z., Bajec, M., Krisper, M.: Enterprise modelling with UML. Electrotech. Rev. **68**, 109–114 (2001)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Marien R. Krouwel is an Enterprise Architect and Software Engineer with over 10 years of experience in the low code domain. He is specialized in creating future-proof and flexible software that truly supports the business and its dynamics. Marien applies principles from Enterprise Design and Engineering to capture business ideas in enterprise models, to subsequently convert these models into smart software solutions. He has experience in both enterprise modelling and software development, and with DEMO and Mendix in particular. After having worked at Capgemini for 12 years, Marien started his own company and co-founded Trives Solutions, aimed at supporting business opportunities by bridging the gap from idea to software solution in short time and in a transparent way. Currently, he acts as independent consultant, enterprise solution architect and trainer. In 2023, Marien received his PhD on the topic of Enterprise Model-Driven Software Development. He frequently supervises MSc research and regularly provides lectures at the Antwerp Management School and TU Wien.

Martin Op 't Land works at Capgemini as a certified Global Architect with over 35 years of experience. He has specialized in chain collaboration and information management, focusing on coherence and evolvability of enterprises. From 2018, he participates in the Architecture Council of the Dutch Criminal Justice System with as his personal mission to contribute to Law Execution as intended. As a lead architect, he prefers to work in complex projects and issues that have not been solved before. People who know him describe him as: curious, asking clarifying questions and working thoroughly. Driven to build meaningful relations, he enjoys working with young and experienced talent bringing coherence and movement. He loves lecturing and doing and supervising research as Professor of Enterprise Engineering at Antwerp Management School and University of Antwerp. As Editor of the Springer Enterprise Engineering Series and Faculty of the EE-network, he contributes to the academic development of Enterprise Engineering, and as active member of the Enterprise Engineering Network, he devotes himself to its practical applicability.

Henderik A. Proper, Erik for friends, is Full Professor in Enterprise and Process Engineering in the Business Informatics Group at the TU Wien. Erik has a mixed background, covering a variety of roles in both academia and industry. His core research drive is the development of theories that work. In other words, Erik focuses on research that leads to results that have both theoretical rigour and practical relevance. His general research interest concerns the foundations and applications of domain modelling, in particular in the context of enterprises. Over the past 20 years, he has applied this research drive and general research interest towards the further development of the field of enterprise engineering, and enterprise modelling in particular. Presently, Erik is vice-chair of the IFIP 8.1 working group, while also being the representative for the Netherlands in IFIP's TC8 technical committee. He is also the Stellvertretender Sprecher of the EMISA working group of the German Computer Science Society (Gesellschaft für Informatik), as well as a member of the management team of the Enterprise Engineering Network.