Generating Low-Code Applications from Enterprise Ontology

Marien R. Krouwel[1,2][0000−0003−4115−3858], Martin Op 't Land[1,3][0000−0003−0024−5908], and Henderik A. Proper[4,5][0000−0002−7318−2496]

[1] Capgemini Netherlands, PO Box 2575, 3500 GN Utrecht, the Netherlands
Marien.Krouwel@capgemini.com, Martin.OptLand@capgemini.com
[2] Maastricht University, Maastricht, the Netherlands
[3] Antwerp Management School, Boogkeers 5, 2000 Antwerp, Belgium
[4] Luxembourg Institute of Science and Technology, Luxembourg
[5] TU Wien, Vienna, Austria e.proper@acm.org

Abstract. Due to factors such as hyper-competition, increasing expectations from customers, regulatory changes, and technological advancements, the conditions in which enterprises need to thrive become increasingly turbulent. As a result, enterprise agility, or flexibility, becomes an increasingly important determinant for enterprise success.
Since Information System (IS) development often is a limiting factor in achieving enterprise flexibility, enterprise flexibility and (automated) IS flexibility cannot be viewed separately and choices that regard flexibility should not be left to developers. By taking a Model-based Engineering (MBE) approach, starting from ontological models of the enterprise and explicit organization design decisions, we bridge the gap from organizational flexibility to (automated) IS flexibility, in such a way that IT is no longer the limiting factor for enterprise flexibility.
Low-code technology is a growing market trend that builds on MBE concepts. In this paper, we report on a mapping for (the automation of) the creation of a Mendix low-code application from the ontological model of an enterprise, while also accommodating the required organizational implementation flexibility. Even though the algorithm has been tested successfully on multiple cases, supporting all possible organizational flexibility seems to be an $\mathcal{NP}$-hard problem, and more research is required to check the feasibility and usability of this approach.

Keywords: Enterprise Ontology · DEMO · MBE · Low code · Mendix.

1 Introduction

Due to factors such as hyper-competition [7], increasing expectations from customers, regulatory changes, and technological advancements, the conditions in which enterprises need to thrive become increasingly turbulent. As a result, the ability to change with an ever decreasing time-to-market, often referred to as 'agility' [32], becomes an important determinant for the success of enterprises [34] – with enterprise we mean every goal-oriented cooperative or network of actors [13], including government agencies, commercial companies, etc.

The notion of 'agile enterprise' is also referred to as the 'run-time adaptive enterprise' [25] or the 'flexible enterprise' [41]. One can, actually, identify different flavors of flexibility that can be considered as cornerstones in creating a flexible enterprise, including strategic flexibility, organizational flexibility, financial flexibility, marketing flexibility, manufacturing flexibility and (automated) Information System (IS) flexibility [41, Fig. 1.4].

Since an IS supports (the implementation of) an organization, the organization and its IS are in fact intrinsically intertwined [13, p. 251]. As a consequence, organizational flexibility and (automated[6]) IS flexibility can not be viewed separately. Organization design decisions are often hard-coded into ISs [2] – something the authors recognize in their current practice quite frequently[7]. This essentially leaves it up to developers to make the right choices to ensure the IS supports the required organizational flexibility. Therefore it is necessary to make the organization design decisions explicit and to make transparent how they are implemented in Information Systems.

The field of enterprise engineering connects the organizational and IS perspective. DEMO (Design and Engineering Methodology for Organizations) is a leading method within the discipline of enterprise engineering [12], with strong methodological and theoretical roots [13, 35]. At the same time, there is an increasing uptake of DEMO in practice, illustrated by the active community and certification institute[8] as well as the reported cases concerning the use of DEMO [2, 13, 29] and integration with other mainstream enterprise modeling approaches such as ArchiMate [14, 21] and BPMN [6, 16, 28].

DEMO distinguishes between *Enterprise Ontology* and *Enterprise Implementation*. Enterprise Ontology concerns the 'essence' of an organization in terms of products and services delivered. Enterprise Implementation pertains to the organizational structure(s) and (socio-technical) means, including the division of work between human actors and IT solutions, as well as their assignment(s) to the organizational structures. Enterprise Implementation can be expressed in a set of values (choices) in relation to a set of Organization Implementation Variables (OIVs) [22], such as functionary type and organizational unit. Though changes to an Enterprise's ontology do occur, most of the time changes pertain to its Enterprise Implementation [11]. As mentioned above, the authors have observed in practice how organization design decisions – values for some OIVs – are often hard-coded into ISs, essentially leaving it up to developers to determine the involved OIVs and associated design decisions.

According to Brambilla e.a. [3], Model-based Engineering (MBE)[9] is a model-based approach for the creation of (automated) ISs that promises to bridge the

[6] In the remainder of this paper, whenever we refer to IS, we do so in the sense of being an automated Information System using Information Technology (IT).

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

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

[9] For this article we consider Model-driven Software Engineering (MDSE), Model-driven Development (MDD), Model-based Development (MBD) and Model-Driven Enterprise Systems Engineering (MDESE) as being specializations of MBE.

gap between requirements of an organization and technical implementation of the IS. Claimed advantages over traditional software development approaches include *a*) common and better understanding of the IS, *b*) increased productivity of the development team due to (partial) automation of the development process, and *c*) reducing the number of defects. In order to be able to execute/transform a model, the model must be comprehensive – describing all business requirements – and consistent – free from contradictions – and must have its semantics fully specified. MBE can involve combinations of code generation from and interpretation of models.

As ontological enterprise models claim to be coherent, comprehensive, consistent and concise [13, p. 14], it seems a good starting point for an MBE approach. Practice already showed that DEMO's ontological models have proven to be a solid starting point for model-based IS design [15, 19, 29] and code generation [20, 23, 24] in particular. However, to the best of our knowledge, there is no complete mapping from the full DEMO metamodel to any software implementation model. Also, existing research seems to neglect the organization implementation aspects (i.e. the OIVs), still resulting in the need for developers to deal with these aspects themselves, thereby not making the mapping from implementation decision to software transparent. For instance, Van Kervel [20] reports on a software engine able to generate working operational software, taking the ontological enterprise model of a domain as input. However, in that approach all choices in terms of organizational implementation remain implicit and end up being hard-coded by choices in the software engine.

A potential explanation for the latter issue 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 we, in principle, suggest using a model-based code generation strategy, as 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. This is where we suggest to turn to low-code technology.

The idea of low-code technology is to add an(other) abstraction layer on top of (high) code such as Java and .Net. Low-code technology applies techniques from MBE, while still allowing for (controlled) customizations and extensions [5]. While low-code technology improves enterprise flexibility, compared to traditional code [39], practical experience shows that these platforms mainly offer IS technical implementation flexibility – for instance changing the database management platform from e.g. MySQL to Postgres, or moving from one front-end framework to another – and some flexibility on the IS (functional) requirements level – for instance changing a specific process flow or screen lay-out. At the same time, however, changes in the organization, such as the organizational structures or its portfolio products and services, still can take quite some time to implement.

Goal and approach. In the research reported on in this paper, we hypothesize that by taking an MBE approach towards low-code, starting from enterprise ontology, one can improve enterprise flexibility even more, at least to the point that IT is not the limiting factor anymore. More specifically, this paper reports on the design of an (automated) mapping to create a Mendix application model from Enterprise Ontology and OIVs. We choose Mendix as low-code platform because it provides good documentation about its metamodel and offers an SDK (model API) to create Mendix applications using TypeScript. Next to that, the authors are experienced in using the Mendix platform in real-world situations.

In line with this, the overall goal of the research reported on in this paper is to create an automatic translation ('mapping') from the ontological enterprise model (cf. DEMO Specification Language (DEMO-SL) [9]) to a Mendix application (cf. the Mendix metamodel[10]) in order to be able to generate low-code applications from ontological (enterprise) models. As this mapping is a designed artifact, we follow the Design Science methodology [40]. In terms of design science, the main focus of this paper is on the design cycle; i.e. the construction of the mapping. The mapping has been evaluated multiple times involving different DEMO models, ranging from academic cases such as EU-Rent [31, 33] to real-world cases including Social Housing [24, 25]. While the mapping has evolved and improved over time, this paper presents the final version of the mapping.

The remainder of this paper is structured as follows. In section 2 we outline the theoretical background on Enterprise Ontology, Enterprise Implementation, MBE and low-code technology. In section 3 we discuss the mapping, while section 4 provides a summary of the evaluations conducted so-far. We finish with conclusions and a discussion of the limitations in section 5.

2 Theoretical Background

The discipline of Enterprise Engineering encompasses Enterprise Ontology, Enterprise Architecture, Enterprise Design and Enterprise Governance [13]. Building upon the Language/Action Perspective [43], based on Speech Act Theory [1, 38] and Theory of Communicative Action [18], it sets communication as the primal notion for the design of enterprises and its Information Systems.

As mentioned in the introduction, DEMO is a leading method within the discipline of Enterprise Engineering, that has been applied at several enterprises. Reported benefits of DEMO include: *a*) ensuring completeness and helping to find omissions and ambiguous situations [13, 29], *b*) providing a solid, consistent and integrated set of (aspect) models (see below) [13], *c*) creating a shared understanding [29], *d*) providing a stable base to discuss required flexibility [2], *e*) offering a good Return On Modeling Effort [13], and *f*) offering a good basis to define IS requirements and design or generate such systems [13].

This section will briefly introduce the most important aspects of Enterprise Ontology and Enterprise Implementation, as well as the concepts of MBE and low-code technology for the IT implementation of Enterprises.

[10] https://docs.mendix.com/apidocs-mxsdk/mxsdk/understanding-the-metamodel/

2.1 Enterprise Ontology

An ontology is defined as a conceptual specification describing knowledge about some domain [17]. More specifically, Enterprise Ontology describes both the dynamics and statics of an enterprise, seen as a network of actors that enter into and comply with commitments [13]. Such commitments are raised by actors in acts, the atomic units of action, and follow the Complete Transaction Pattern (CTP). The CTP 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 idea is that actors constantly check whether there are acts they have to deal with or respond to. The total set of acts for an actor to deal with is called the actor's *agenda*.

By abstracting actors to actor roles the model becomes independent of the people involved in the operation. Commitments regarding a specific product kind are abstracted to transaction kinds, all having one actor role as executor and one or more actor roles as initiator. An Enterprise Ontology is expressed in a set of aspect models that is comprehensive, coherent, consistent, and concise [13]. The reader is referred to [9] for the complete DEMO metamodel.

Ontological Aspect Models. The ontological model of an organization consists of an integrated whole of four aspect models [13]. The *Cooperation Model (CM)* models the *construction* 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 *Process Model (PM)* models the *processes* (dynamics) 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 *Fact Model (FM)* is the semantic model of *products* (statics) of the enterprise; it defines (elementary or derived) fact types (entity types with their related product kinds, property types, attribute types and value types), existence laws and occurrence laws. The *Action Model (AM)* is the model of the *operation* of the enterprise, guiding actors in performing their acts; guidelines consist of an *event part* detailing the act to respond to, an *assess part* detailing the conditions to check and a *response part* stating how to respond. Note that these guidelines look like rules, but offer the actors the possibility to – responsibly – deviate.

2.2 Enterprise Implementation

In order for an organization to be operational, it has to be implemented with appropriate technology [13]. When a product or service is decided upon, and the collaboration network needed for its delivery has been revealed in the ontological (enterprise) model, still many degrees of freedom exist before an organization can become operational [13, p. 349]. Implementation design starts from the (enterprise) ontology of a system and ends with a fully detailed specification of the implementation, within the design space constrained by Enterprise Architecture [10]. The lowest level and most detailed model of an organization describes

all implementation design decisions, including organizational structure(s), the decision to automate certain tasks, as well as the assignment of parties, people as well as IT and other means to the organizational structures [22].

The notion of OIV expresses a category of design choice(s) for organizational implementation [22]. It is a variable in the sense that its value can be different for different implementation alternatives. Some examples of the ∼25 of such OIVs [22] include:

- deciding upon *functionary types* and which actor roles or acts they should fulfill – e.g., the functionary type 'cook' that fulfills both the actor roles 'baker' and 'stock controller'; or the functionary type 'deliverer' who is authorized for the acts promise and declare of the (transaction kind) transport and also for the act of accept for the customer payment;
- deciding on *order of working* – e.g., should delivery only be done after receiving payment (as common in retail) or is it (also) possible to pay afterwards (more common in B2B); and
- deciding on *work locations & organizational units*, e.g., which branches exist, and what functionary types do exist there.

Implementations of organization, and therefore the value of OIVs, will change far more often than the products / services they help deliver. E.g., actor roles as 'sales' and 'baker' are stable notions in a pizzeria, just as the act 'accept payment'; what might change often is the authority of a functionary type – answering questions such as 'shall we combine the actor roles of sales and baker in one functionary type in the first months of opening our new pizzeria branch?' or 'shall we take the responsibility for accepting payments away from our functionary type deliverer, now payment accepting is executed automatically by a web agent under the (outsourced) responsibility of a payment service provider?'.

For a flexible enterprise it is a priority that frequently occurring changes, typically implementation choices, are not on its critical path; therefore ideally it should be possible to make such changes with no or only little impact in IS. However, organizationally variability is so large that conscious choices have to be made what organizational changes should be supported by the IS to what extent. Even the simplistic assumption of each of the 25 OIVs being able to independently change with 3 values each already creates $3^{25} \approx 8.5 * 10^{11}$ different organizational implementations for each transaction kind – a problem for automation, but even more so for human overview. OIVs make it possible to make explicit the organization design decisions that need to get a place in IS. It also provides the possibility to make an explicit requirement that the values should be (easily) changeable in the IS.

2.3 Model-based Engineering and Low-code Technology

Model-based Engineering is an approach to application development where models are transformed to (high) code by means of code generation or model interpretation [3]. While there are differences between code generation and model

interpretation in terms of easiness to understand, debugging possibilities, performance of the execution environment, and compile and deploy time, from a usage perspective these difference do not really matter; both need a mapping from higher order (business domain) model to lower order software model, and they can even be combined [4].

The term low code was introduced by Forrester in 2014 [36]. Low-code development platforms enable quick creation of software with minimum efforts compared to traditional procedural computer programming [42] – also known as 'high code', such as Java and .Net. It builds upon existing concepts including MBE, code generation and visual programming [8]. Claimed benefits of low code include [26,37,39] *a*) less hand-coding, faster development, and, as a result, cost reduction in both development and maintenance and a shorter time-to-market, *b*) complexity reduction by using prebuilt components, *c*) the ability for non-technical people to create applications, thus opening up the possible population for application development as well as improving business and IT collaboration while increasing IT productivity, and *d*) enabling digital transformation and increasing IT and business agility.

Low-code platforms, including Mendix, rely on 3 basic concepts: *data* (in Mendix: Entity, Enumerations, Attribute and Association), *logic* (also called action or (micro)service; in Mendix: Microflow) and *interface* (Application Programming Interface (API) and screen; in Mendix: Page, containing data views and/or Buttons), as well as their interrelations and (Access) Rules for User Roles enabling users with certain roles to use these parts of the application. Due to page limitations, the reader is referred to footnote 10 for the complete Mendix metamodel.

3 Mapping

The mapping is based on DEMO-SL [9] and we included all concepts relevant for IT implementation. As the current version of DEMO-SL only describes the metamodel of Enterprise Ontology, we added the concept of Organization Implementation Variable. The resulting mapping can be found in Table 1. Some of the design decisions regarding the mapping include:

D.1 Transaction kinds are not mapped. Instead, Product kinds or Event types are mapped to an Entity in order to be able to capture the state of a transaction.

D.2 For aggregate transaction kinds (ATKs) it is usually not needed to capture the coordination acts around these facts, so no mapping is needed. The production facts in the ATK are present in the FM and thus mapped to a Mendix unit, see also D.4

D.3 As the page for showing the agenda for an actor is very generic functionality, we choose not to generate it but built it in Mendix as a reusable component. The logic to support the state machine representing the Complete Transaction Pattern is also built as a generic module. Note that the details of this module are not part of this paper.

DEMO concept (aspect model)	example	Mendix unit
Elementary transaction kind (CM)	TK01	n/a, see D.1
Aggregate transaction kind (CM)	MTK01	n/a, see D.2
Actor role (both elementary and aggregate) (CM)	AR01	User Role, see D.3
Executor link (CM)	AR01-TK01	Action Button and (Microflow) Access Rule
Initiator link (CM)	CAR01-TK01	Action Button and (Microflow) Access Rule
Access link (CM)	CA01-MTK01	Entity Access Rule
Product kind (CM)	[registration] is started	n/a, see Event type
Transaction general step kind (PM)	TK01/rq	Page
non-derived Entity type (FM)	Registration	Entity and Pages, see D.4
aggregation Entity type (FM)	{Registration X Year}	Entity with Associations to its aggregates
specialization Entity type (FM)	Started Registration	n/a, see D.5
generalization Entity type (FM)		n/a, see D.5
Property type (FM)	member	Association
Attribute type (FM)	start date	Attribute or Association, see D.6
Value type (FM)	day, money	Enumeration or Entity, see D.6
Event type (FM)	[registration] is started	Entity having $Transaction.Proposition$ as generalization, with Association(s) to its variable(s), see D.7
Derived fact kind (FM)	age	Microflow, see D.8
Action rule-event part (AM)		Action Button and (Microflow) Access Rule, see D.9
Action rule-assess part (AM)		Microflow and Page, see D.9
Action rule-response part (AM)		Action Button, Microflow and (Microflow) Access Rule, see D.9
Organization Implementation Variable	functionary type	Entity, see D.10

Table 1. Mapping from DEMO metamodel (concept) to Mendix metamodel (unit)

D.4 For external 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 Data Hub. For now we decided to not use that possibility, especially as this does not seem possible (yet) through the Mendix SDK. Instead, we will create some basic CRUD (Create, Read, Update, and Delete) pages to modify and view the data. It is fairly easy in the generated application to change this later.

D.5 We have not seen enough DEMO models to provide a mapping for the generalization and specialization concepts.

D.6 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 an Enumeration or to an Entity. The Attribute Type using the Value Type will then either be an EnumerationTypeAttribute or an Association. If the scale sort of the Value Type is of some other type (day, money, etc.) the Attribute Type will be mapped to some AttributeType (DateTime, Decimal, etc.).

D.7 *Transaction.Proposition* is an Entity that is part of the generic module handling the CTP. By extending it, we can use the generic state machine, but also relate it to the specific entity or entities the Event type is about.

D.8 Derived fact kinds need to be calculated. Currently we suggest to create a microflow in order to do that. Its (mathematical) definition needs to be implemented in the microflow, preferably this mapping is also provided but we consider this too detailed for the scope of this paper. A decision that goes along with this is that from a performance perspective one would like to decide whether this calculation is performed on read or on save. The low-code approach we take makes it easy to make such a decision in the platform, as it currently seems to difficult to insert that aspect into the mapping.

D.9 For the handling of Action rules, we decided to implement this as an Action Button for the user role that has to deal with the agendum (type), a Page to see all the relevant information to decide on a response, as well as one or more Action Buttons for the different choices. In this way we respect the autonomy of the actor(s) involved, and only automate the parts for retrieving all the information. The detailing of the assess-part is similar to that of a derived fact kind and a similar reasoning as D.8 holds.

D.10 OIVs are considered to be adaptable at run-time, and are therefore mapped to an Entity. The different values of such an OIV can have impact on authorization, redirecting and handling of an act, and much more. This is considered to be part of the logic and state machine and supports our choice for low code as target platform because we consider it is easier to build it into the state machine than into the translator. A more elaborate mapping can therefore not be provided, this mapping at makes it possible to change the (values of the) OIVs in the running application. In subsection 2.2 we already showed that the possible number of configurations grows exponentially with the number of OIVs and we therefore think incorporating OIVs into the logic might turn out to be a $\mathcal{NP}$-hard problem.

4 Implementation and evaluation

As part of the design cycle of Design Science, we have evaluated the mapping on several cases, ranging from academic cases such as EU-Rent [31,33] to real-world cases including Social Housing [24,25]. At first, this mapping was performed manually, while later we created a TypeScript[11] implementation of the mapping[12], that has been evaluated, improved and extended to deal with additional concepts and cases by executing it with different DEMO models as input. Newer cases did not result in major redesigns of the mapping.

While there is an XML-based exchange model for DEMO available [30], we decided to use a more compact JSON format to represent the DEMO metamodel including the concept of OIV, while leaving out parts that are for representation only, such as diagrams and tables. As an illustration, building upon the Social Housing case described in [25], Figure 1 shows the input (JSON) files for the automated translator, while Figure 2 shows the project folder and domain (data) model of the generated Mendix application[13].

descriptor.json

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

demomodel.json

```
{
    "transactionkinds": [
        {"id": "TK01", "name": "registration starting", "type": "elementary"},
        {"id": "TK02", "name": "registration paying", "type": "elementary"}
    ],
    "actorroles": [
        {"id": "CAR01", "name": "(aspirant) member", "type": "composite"},
        {"id": "AR01", "name": "registration starter", "type": "elementary"}
    ],
    "factkinds": [
        {"name": "Registration", "type": "entitytype"},
        {"name": "Person", "type": "entitytype"},
        {"name": "Nationality", "type": "valuetype", "values": "NL, EN"},
        {"name": "member", "type": "propertytype", "domain": "Registration", "range": "Person"},
        {"name": "payer", "type": "propertytype", "domain": "Registration", "range": "Person"},
        {"name": "starting day", "type": "attributetype", "domain": "Registration", "range": "datetime"},
        {"name": "day of birth", "type": "attributetype", "domain": "Person", "range": "datetime"},
        {"name": "nationality", "type": "attributetype", "domain": "Person", "range": "Nationality"},
        {"name": "StartedRegistration", "type": "eventtype", "parameter": "Registration", "product": "registration is started"},
        {"name": "EndedRegistration", "type": "eventtype", "parameter": "Registration", "product": "registration is ended"},
        {"name": "PersonAge", "type": "derived", "parameter": "person"}
    ],
    "actionrules":
    [
        {"actorrole": "AR01", "name": "T01rq"}
    ],
    "oivs": [
        {"name": "functionary type"},
        {"name": "organizational unit"}
    ]
}
```

Fig. 1. Descriptor file and DEMO model in JSON format for Social Housing

[11] TypeScript is the language to access the Mendix SDK, see https://docs.mendix.com/apidocs-mxsdk/mxsdk/

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

[13] More details on the case as well as screen shots can be found in [25]

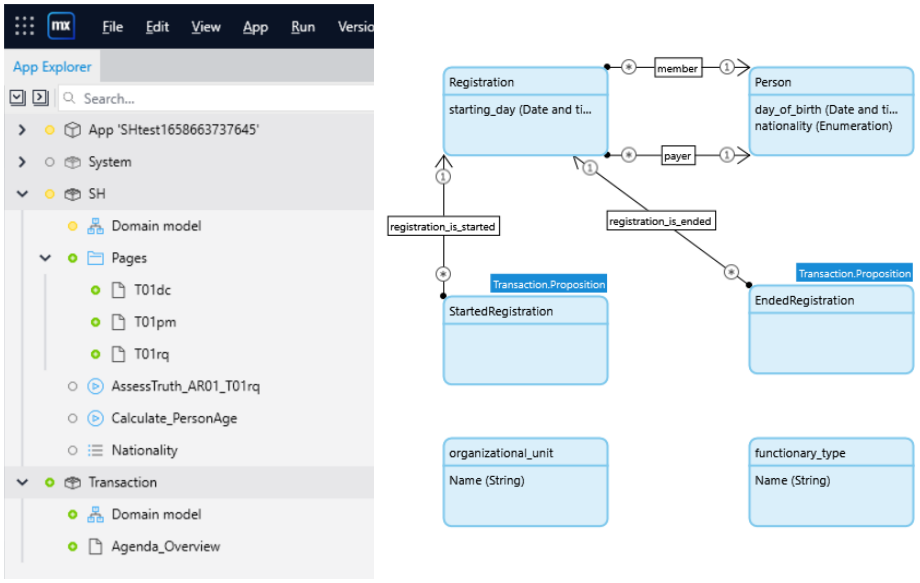

Fig. 2. Project outline and domain model of the generated Mendix application for Social Housing

By introducing the concept of Organization Implementation Variable into the MBE process or code generation, we were able to make transparent the mapping from organizational design decision to software implementation, thereby not leaving it up to developers to make the right choice. As the Enterprise Implementation is implemented as a configurable unit in terms of OIVs, it is easy to change the Enterprise Implementation for a given Enterprise Ontology, thereby improving the enterprise flexibility, as was shown in earlier research [25].

5 Conclusions and Discussion

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

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

It also adds flexibility as parts of the IS are simply (re)generated when new transaction kinds arise. By adding OIVs to the MBE approach, it becomes pos-

sible to change (some) design decisions at run-time, therefore allowing for even more flexibility and agility for the enterprise.

Another advantage of this approach seems to be that the generated code can easily be adapted through the low-code visual paradigm. This allows for changes in the UI, to make use of APIs, or to implement the execution of the action rules or calculations in a more efficient way. A warning should be given that changes in the generated code can become a source of hidden design decisions.

Reflecting on our MBE approach, by making explicit the required organizational flexibility and giving it a specific place in the generated code, we have successfully connected organizational flexibility and IS flexibility, thereby at least improving enterprise flexibility. As the code can easily be regenerated, changing the IS is not the limiting factor anymore in changing the enterprises service or product portfolio or its implementation.

Limitations and Future Research Recommendations

As the details of the generic component supporting the CTP do not fit into this paper, it has to be described and published separately.

In creating the mapping, we noticed the DEMO metamodel in DEMO-SL is not specified from an operational point of view but from a diagramming perspective. In the (manual) conversion to JSON, we also had to add aspects that were relevant for generating software, such as the application and module name. It seems necessary to better detail the generic metamodel, containing only the true DEMO concepts and separating the core of DEMO from the purpose-specific information such as visualization or code generation.

For the implementation of an external entity type, such as person, in the FM, one typically chooses to use an existing source (through an API) or to create an own database table to store the relevant data (incl. Create, Read, Update, and Delete (CRUD) pages). The current version of the mapping does not yet allow for this differentiation, partly as a result of lack of support through the SDK. A future version of the translator should incorporate this. Note that by the nature of low-code, it is always possible to change the output model to incorporate the usage of existing APIs.

Mendix has launched native workflow capability in the platform. As the majority of the mapping and its implementation was set up before that time, it doesn't 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 code generator.

The implementation of OIVs in Mendix, or maybe even software in general, is not straightforward. The impact of certain choices on the state machine and logic is not completely detailed yet. This will differ per OIV and can only be detailed per individual OIV. It can even be that when combining different OIVs, the problem becomes to complex to solve. It is not clear whether all OIVs can be implemented completely independent of others, as suggested by Normalized Systems theory [27]. We will need more evaluations with more cases involving more

OIVs to fully understand the complexity of this problem. Supporting all possible organization design decisions could even be an $\mathcal{NP}$-hard problem. Further research is required to check the feasibility and limitations of this approach.

The mapping we made is specific towards the Mendix platform. Although other low-code platforms rely on similar concepts, the question arises whether the mapping can be abstracted to facilitate other low-code platforms, or even high code. Further research is needed to get a perspective on the feasibility and usability of such an abstraction.

It's hard to compare our MBE approach to ones that use a different kind of input, such as BPMN or UML. It could be interesting to research ways to compare the advantages and disadvantages of different approaches towards MBE.

References

1. Austin, J.L.: How to do things with words. William James Lectures, Oxford University Press (1962)
2. Bockhooven, S.v., 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 (July 2015), http://ceur-ws.org/Vol-1408/paper3-tee.pdf
3. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers (09 2012). https://doi.org/10.2200/S00441ED1V01Y201208SWE001
4. Cabot, J.: Executable models vs code-generation vs model interpretation. Online (Aug 2010), https://modeling-languages.com/executable-models-vs-code-generation-vs-model-interpretation-2/, accessed 2022-May-17
5. 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
6. 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 (Jun 2012). https://doi.org/10.1201/b12295-17
7. D'aveni, R.A., Gunther, R.: Hypercompetition. Free Press (Mar 1994)
8. 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? Software and Systems Modeling **21** (01 2022). https://doi.org/10.1007/s10270-021-00970-2
9. Dietz, J.: The DEMO Specification Language v4.7. Tech. rep., Enterprise Engineering Institute (Apr 2022), https://ee-institute.org/download/demo-specification-language-4-7-1/
10. Dietz, J.L.G.: Architecture – Building strategy into design. Netherlands Architecture Forum, Academic Service – SDU, The Hague, The Netherlands (2008)
11. Dietz, J.L.G., Hoogervorst, J.A.P.: Enterprise Ontology and Enterprise Architecture – how to let them evolve into effective complementary notions. GEAO Journal of Enterprise Architecture, 2007 **1** (2007)

12. 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. International Journal of Organisational Design and Engineering **3**(1), 86–114 (2013). https://doi.org/10.1504/IJODE.2013.053669
13. 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
14. Ettema, R., Dietz, J.L.G.: ArchiMate and DEMO – Mates to Date? In: Albani, A., Barjis, J., Dietz, J.L.G., Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C. (eds.) Advances in Enterprise Engineering III, Lecture Notes in Business Information Processing, vol. 34, pp. 172–186. Springer Berlin Heidelberg (2009). https://doi.org/10.1007/978-3-642-01915-9_13
15. Falbo, R., 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) (Jun 2002)
16. 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
17. Guizzardi, G.: Ontological foundations for structural conceptual models. Ph.D. thesis, University of Twente (Oct 2005)
18. Habermas, J.: The theory of communicative action. Cambridge: Polity Press (1986)
19. de Jong, J.: A Method for Enterprise Ontology based Design of for Enterprise Information Systems. Ph.D. thesis, TU Delft (2013)
20. van Kervel, S.: Ontology driven Enterprise Information Systems Engineering. Ph.D. thesis, TU Delft (2012)
21. Kinderen, S.d., Gaaloul, K., Proper, H.A.: On transforming DEMO models to ArchiMate. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Wrycza, S. (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
22. Krouwel, M.R., Op 't Land, M., Offerman, T.: Formalizing Organization Implementation. In: Advances in Enterprise Engineering X. pp. 3–18. EEWC 2016, Funchal, Madeira Island, Portugal (2016). https://doi.org/10.1007/978-3-319-39567-8_1
23. Krouwel, M., Op 't Land, M.: Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems. In: Albani, A., Dietz, J.L.G., Verelst, J. (eds.) Advances in Enterprise Engineering V (EEWC-2011). Lecture Notes in Business Information Processing, vol. 79, pp. 31–45. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-21058-7_3
24. Krouwel, M.R. and Op 't Land, M: Business Driven Micro Service Design - An Enterprise Ontology based approach to API Specifications. In: Aveiro, D., Proper, H., Guerreiro, S., de Vries, M. (eds.) Advances in Enterprise Engineering XV. vol. 441. Springer (2021). https://doi.org/10.1007/978-3-031-11520-2_7
25. Op 't Land, M., Krouwel, M., 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: Aveiro, D., Guizzardi, G., Pergl, R., Proper, H.A. (eds.) EEWC 2020. pp.

228–242. No. XIV in Advances in Enterprise Engineering, Springer (Apr 2021). https://doi.org/10.1007/978-3-030-74196-9_13
26. Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J.: Characteristics and challenges of low-code development: The practitioners' perspective. CoRR (2021), https://arxiv.org/abs/2107.07482
27. Mannaert, H., Verelst, J.: Normalized systems: re-creating information technology based on laws for software evolvability. Koppa, Kermt, Belgium (2009)
28. Mráz, O., Náplava, P., Pergl, R., Skotnica, M.: Converting demo psi transaction pattern into bpmn: A complete method. In: Aveiro, D., Pergl, R., Guizzardi, G., Almeida, J.P., Magalhães, R., Lekkerkerk, H. (eds.) Advances in Enterprise Engineering XI. pp. 85–98. Springer International Publishing, Cham (2017)
29. Mulder, J.B.F.: Rapid Enterprise Design. Ph.D. thesis, Delft University of Technology (2006)
30. Mulder, M.: Enabling the automatic verification and exchange of DEMO models. Ph.D. thesis, Radboud University Nijmegen (2022), https://repository.ubn.ru.nl/handle/2066/247698
31. Object Management Group: Business Motivation Model. Tech. Rep. Version 1.3, Object Management Group (May 2015), http://www.omg.org/spec/BMM/1.3/PDF/
32. Oosterhout, M.P.A.v.: Business Agility and Information Technology in Service Organizations. Ph.D. thesis, Erasmus University Rotterdam (June 2010)
33. Op 't Land, M., Krouwel, M.R.: Exploring Organizational Implementation Fundamentals. In: H.A. Proper, D.A., Gaaloul, K. (eds.) 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
34. Overby, E., Bharadwaj, A., Sambamurthy, V.: Enterprise agility and the enabling role of information technology. Eur. J. Inf. Syst. **15**, 120–131 (April 2006). https://doi.org/10.1057/palgrave.ejis.3000600
35. Reijswoud, V.E.v., Mulder, J.B.F., Dietz, J.L.G.: Communicative Action Based Business Process and Information Modelling with DEMO. The Information Systems Journal **9**(2), 117–138 (1999)
36. Richardson, C., Rymer, J.: New Development Platforms Emerge For Customer-Facing Applications. Tech. rep., Forrester (2014)
37. Sanchis, R., García-Perales, Fraile, F., Poler: Low-Code as Enabler of Digital Transformation in Manufacturing Industry. Applied Sciences **10**, 12 (12 2019). https://doi.org/10.3390/app10010012
38. Searle, J.R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge, London (1969)
39. Sijtstra, J.: Quantifying low-code development platforms effectiveness in the Dutch public sector. mathesis, Leiden University (Jun 2022), https://theses.liacs.nl/2221
40. Simon, H.A.: The Sciences of the Artificial (3rd Ed.). MIT Press, Cambridge, MA, USA (1996)
41. Sushil, Stohr, E.A. (eds.): The Flexible Enterprise. Flexible Systems Management, Springer India (01 2014). https://doi.org/10.1007/978-81-322-1560-8
42. 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
43. Weigand, H.: Two decades of language/action perspective. Natural Language Engineering **49**, 45–46 (01 2006)