Towards IT Platform Independence with pimUML

From Semantically Rich DEMO Models to Low Code

Nicholas A. Bzowski[1,*], Marien R. Krouwel[2] and Henderik A. Proper[1]

[1]Business Informatics Group, TU Wien, Vienna, Austria
[2]Make IT Right, Utrecht, The Netherlands

Abstract

With the ever-growing complexity of modern enterprises, and their supporting IT systems, it becomes increasingly challenging to maintain good business-IT alignment. In recent work, we reported on a model-driven engineering approach to transform, (business) semantically rich, DEMO models to low-code software artifacts for the Mendix low-code platform, with the aim to improve business-IT alignment. The latter approach, however, heavily depends on the specifics of the chosen platform. To reduce IT platform dependence, the Model Driven Architecture approach suggests to discern three levels of models of a system: a business-oriented computation independent model (CIM), an (IT) platform independent model (PIM), and an (IT) platform specific model (PSM). In this paper, we present a more refined approach with the aim to increase the extensibility of the existing DEMO to Mendix transformation to other target IT-platforms, while also "opening up" for other CIMs besides DEMO models. The development of this approach is done in multiple (agile) design cycles, in which pimUML, a novel UML profile to express PIM models, is developed and evaluated for preservation of semantics in each transformation step.

Keywords

Model Driven Architecture, low code, enterprise ontology, DEMO, UML, Mendix

1. Introduction

Enterprise computing has evolved from simple automation and inventory control on mainframes in the 1960s into Enterprise (Management) Information Systems that support all day-to-day operations across multiple departments and even across enterprises [1, 2]. This advancement has resulted in the role of IT becoming increasingly intertwined with corporate strategy and operations [2, 3], while at the same time business environments are constantly faced with both opportunities and threats as a result of competition, increasing customer expectations, changing regulations, and emerging technologies [4]. Consequently, it becomes even more important that enterprises are able to simultaneously adapt business and IT, with *business-IT alignment* as a critical success factor [5, 6].

Due to the complexity of modern Enterprise Information Systems, maintaining business-IT alignment can be challenging. One solution to overcome this gap can be found in generating software artifacts from business-oriented models [7], also known as Model-Based (Systems) Engineering [8] or Model-Driven Software Development [9, 10], of which low code can be considered a more recent implementation [11]. A popular approach includes Model Driven Architecture®[1] [12] (MDA), in which higher-level enterprise models can be transformed to code through a series of model-to-model transformations (see Fig. 1). MDA suggests discerning three levels of models of a system: a business-oriented computation independent model (CIM), an (IT-)platform independent model (PIM), and an IT-platform specific model (PSM). However, MDA itself does not prescribe a certain level of business semantics of the models involved, nor can it guarantee that the transformations preserve all semantics [13].

In recent work [15, 6], we explored the possibilities of mapping (business) semantically rich DEMO

PoEM2025: Companion Proceedings of the 18th IFIP Working Conference on the Practice of Enterprise Modeling: PoEM Forum, Doctoral Consortium, Business Case and Tool Forum, Workshops, December 3-5, 2025, Geneva, Switzerland

*Corresponding author.
nbzowski@gmail.com (N. A. Bzowski); marien@make-it-right.nl (M. R. Krouwel); henderik.proper@tuwien.ac.at (H. A. Proper)
0009-0002-3814-6876 (N. A. Bzowski); 0000-0003-4115-3858 (M. R. Krouwel); 0000-0002-7318-2496 (H. A. Proper)

[1]Model Driven Architecture is a registered trademark of the Object Management Group

Figure 1: The Model Driven Architecture process, as described in [14]

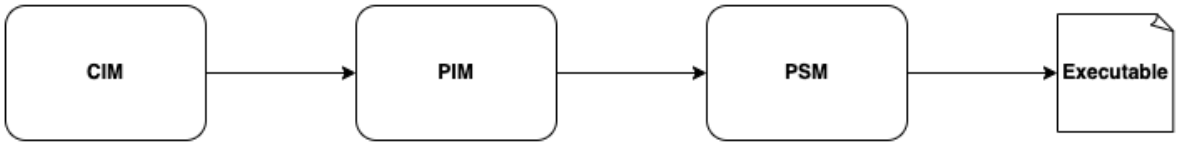
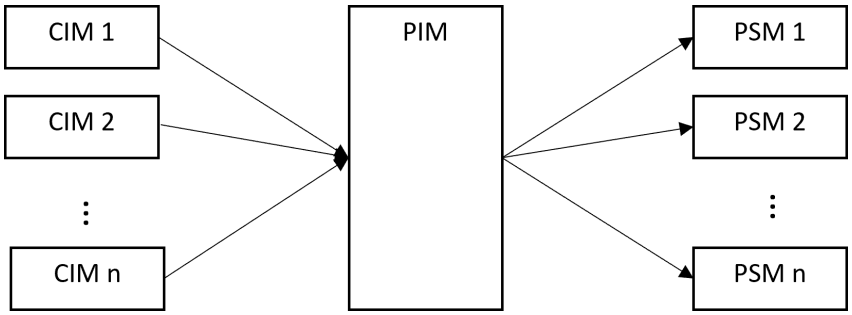


Figure 2: PIM as neutral intermediate model between multiple CIMs and PSMs



enterprise models directly to the Mendix low-code platform. In terms of MDA, this constitutes a direct CIM-to-PSM transformation. By introducing a PIM layer in between, the process can be relatively easily opened up to accommodate other PSMs as well as other CIMs, analogous to a router in a network (Fig. 2). This research aims to improve the extensibility of the existing DEMO to Mendix transformation to other IT-platforms while "opening up" for other (complementary) CIMs besides DEMO.

One promising PIM language is B-UML [16], which was developed in parallel to the research reported in this paper. However, the B-UML language appears to be designed primarily as an intermediate model for code-to-code transformation [16], or as an intermediate model from models with relatively low (business) semantics [17]. Other initiatives include DISME [18]. However, the latter initiative combines DEMO enterprise modeling with platform-specific constructs, completely ignoring the distinction between the three MDA layers.

In this paper, we report on the development of pimUML, a set of UML profiles to express PIM models. In developing pimUML, the transformation mappings from the DEMO Fact Model to pimUML and from pimUML to Mendix have been redefined and evaluated for the level of semantic preservation.

The remainder of this paper is structured as follows. Section 2 is concerned with the research approach we have used in developing pimUML. In Sect. 3 the relevant literature regarding MDA, UML, DEMO, and Mendix are summarized. Finally, the results and evaluation of this research are presented in Sect. 4, after which conclusions and future research are discussed in Sect. 5.

2. Research Approach

As the goal of this research is to define a PIM to allow a stepwise transformation from DEMO to low code, we adopt the design science research methodology (DSRM) for the creation and evaluation of innovative artifacts. As both the problem space and solution space need to be further refined, we adopt a more iterative approach, also known as agile design science (ADSRM; see [19]). The latter approach is based on the generic DSRM, while being augmented with agile techniques for studies in which iterative and experimental design is necessary. In terms of being 'agile', it adds features such as "problem backlog" and "hardening sprint". With each new sprint, findings may give rise to new questions being posed that are added to the problem backlog to be addressed in a later sprint. To maintain rigor, a hardening sprint may be performed periodically to solidify findings from previous sprints.

The actual research project was executed in eight sprints (see Table 1) of three weeks each.[2] The

[2]Not all results could be included in this paper; more details can be found in [20]

Table 1
Overview of the performed sprints and results

Sprint	*Activities*	*Results*
1	Defining problem scope and setting solution objectives	Sect. 3
2	Semi-structured literature review	Sect. 3
3	Metadesign	Sect. 4.1
4	PIM development and evaluation: standard UML	
5	PIM development and evaluation: xUML	
6	PIM development and evaluation: fUML	
7	Hardening sprint: defining pimUML	Sect. 4.2
8	Development and evaluation of mapping from pimUML to Mendix	Sect. 4

first sprint was focused on scoping the problem by analyzing relevant literature. It turned out that a better understanding of the three MDA abstraction layers was necessary, which was consequently tackled in a semi-structured literature review in sprint 2. After a metadesign sprint (3), sprints 4-6 focused on different UML profiles (see Sect. 4.2) to define a PIM and a CIM-to-PIM transformation, all using the procedure from [21]. In the hardening sprint 7, the strongest components of the previous three sprints were synthesized into pimUML, which in sprint 8 was evaluated with the development of a transformation to Mendix. The mappings created during the design sprints were demonstrated and evaluated using the EU-Rent case [22, 23]. To quantitatively evaluate pimUML, the semantic preservation of transformations to and from the PIM was established [24, 25].

3. Theoretical Background

As this research builds upon existing concepts and artifacts, such as MDA, DEMO, UML and low code, this section provides the relevant background.

3.1. Model Driven Architecture

Model Driven Architecture (MDA) is an example of Model Based Engineering (MBE) and Model Driven Software Development (MDSD) (and related terms, see [26]) that uses UML to visualize and generate code [27]. Where MDSD typically uses only one step of model transformation (or code generation), MDA uses a series of model transformation to produce code (see Fig. 1). Typical advantages of model transformations, including MBE, MDSD and MDA, are *a*) better understanding [28, 29, 30], *b*) increased productivity [28, 29, 30], and *c*) traceability between model and code [31, 32] – key to achieve for business IT-alignment.

To better understand the applicability of the MDA approach to generate low-code applications from DEMO models, a semi-structured literature review was conducted (sprint 2) to better understand the three abstraction layers [20]. The resulting (working) definitions of CIM, PIM and PSM are:

CIM – The computation-independent model captures the business operations, including domain-specific vocabulary, actors, processes, and business rules. It focuses on functional aspects and tends to completely stay away from specific technologies [33]. Its primary users are business analysts, enterprise architects, and other business experts [33, 34].

PIM – The platform-independent model describes the computational concerns of applications, primarily by capturing architectural and functional aspects of software systems. There tends to be a focus on human-machine interactions, in both process design and user interface design [35]. A PIM is primarily used by software architects and system analysts [34].

PSM – The platform-specific model may take many different forms, depending on the chosen target technology. It covers many application aspects, from database design to user interface design. Given their technical nature, PSMs are mostly used by software developers and system administrators [34].

For this research, DEMO fits the definition of a CIM, while a Mendix application model contains the information as described for a PSM. We will use a UML-based language to define the intermediate PIM, as UML is most commonly used for that layer. These three models and modeling languages will be further detailed in the next sections.

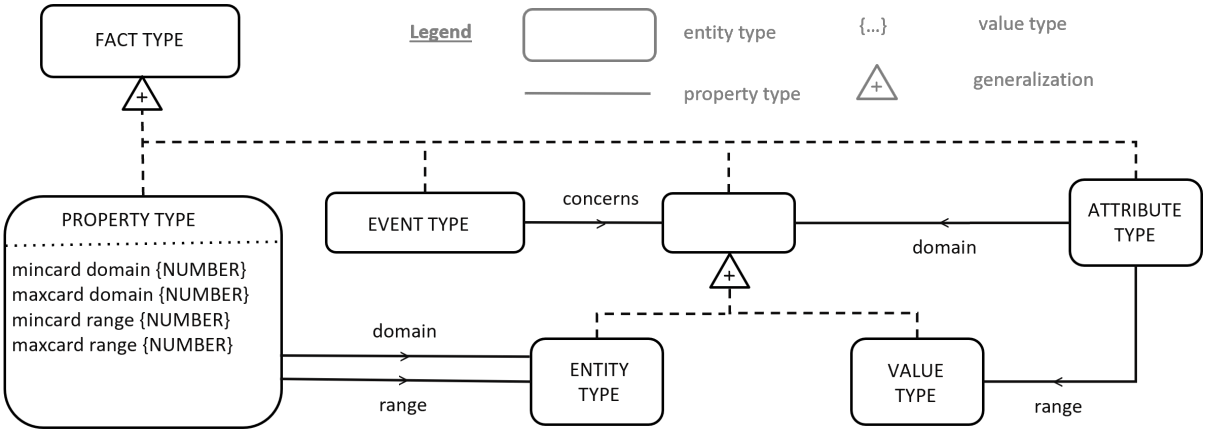
3.2. CIM level: DEMO

The Design and Engineering Methodology for Organizations (DEMO) is a leading method within the discipline of enterprise engineering [36], that sees enterprises as complex sociotechnical systems [37]. It has strong methodological and theoretical roots [38, 39, 40, 41] and sets communication as the primal notion for the design of enterprises and its supporting software systems [42]. A DEMO model aims to capture the operation of an enterprise in a technology-independent way [23] – this is also called the *enterprise ontology* or its *essence*.

One of the key postulates behind DEMO is that facts in the world are created by acts, and that these acts (and associated facts) follow a generic pattern in which coordination – or communication – regards a production (act): the *transaction*. Every transaction goes through this pattern that includes 7 basic steps and a metapattern for cancellations, supporting all possible business exceptions. In every transaction, two actors are involved: one as initiator, interested in the product or service, and one as executor, responsible for delivering the product.

Based on the theories, DEMO includes a modeling language and a modeling procedure to ensure both internal (model) consistency and external consistency (with the real world). DEMO [43] discerns four model aspects: Cooperation (transaction kinds), Process (detailed dependencies), Fact (see below), and Action (business rules). This research focuses on the DEMO Fact aspect that shows the semantic model of products of the enterprise by defining (declared or derived) fact types (entity types with their related product kinds, property types, attribute types, value types, and event types), existence laws, and occurrence laws. Its metamodel is shown in Fig. 3; not shown in the picture is that a fact type is either declared or derived (specialization, generalization, or aggregation).

Figure 3: Metamodel of the DEMO Fact Model, adopted from [43]



3.3. PIM level: UML

The Unified Modeling Language (UML) was chosen as the primary candidate conceptual modeling language for use in the experimental designs at the PIM level for a few key reasons. First and foremost, it is recommended for use as a PIM by the OMG [44]. This is supported by the results of the semi-structured literature review of this study, where UML was found to be the most commonly used PIM language among the studies reviewed. Moreover, UML is the most commonly used conceptual modeling language in the field of software engineering, both in industry and in academia [45]. Another key benefit of UML is its ability to be extended and tailored for particular uses by defining profiles [44].

Although many variants of UML have been defined, for this research, three are considered most relevant: 'standard UML', executable UML, and foundational UML. "Standard UML" is the term that we use to refer to the latest version of UML (at the time of writing: v2.5.1) [46]. Executable UML (xUML), is an executable UML profile proposed by Mellor [47], that prescribes the use of the UML state machine diagram and action language to capture the execution semantics of a domain. Foundation UML (fUML), is an executable subset of UML [48], similar to xUML with a focus on capturing execution semantics, but prescribing activity diagrams for (graphically) expressing behavior and using the Alf action language for a precise definition of the execution semantics.

3.4. PSM level: Mendix (low code)

Low code is a software development approach that allows developers to visually build applications, primarily by using graphical editors with drag-and-drop functionality, with minimal manual coding required [49, 50, 51]. Low code builds on MDSD principles and applies a higher level of abstraction compared to high code, with main advantages including faster time-to-market and increased business agility [52]. Low-code use cases can be found across different lines of business, where the applications range from rapid prototyping to complete digital transformations [49, 53]. Despite their benefits, the two most mentioned challenges in the adoption of low code regard the learning curve and the risk of technology (or vendor) lock-in [50, 53, 52].

Mendix[3] is a low-code application development platform that is currently owned and maintained by Siemens. A Mendix application consists primarily of four different components that realize the application architecture: domain (data) models, pages (user interface), micro- and nanoflows (server and client based logic resp.), and workflows (long-running application processes).[4] Mendix can be deployed on different cloud environments and supports several types of databases.

4. Results

The results of this research include a metadesign, pimUML, and the (updated) transformation mappings, along with a demonstration and evaluation.

4.1. Metadesign

Based on the results of the semi-structured literature review on the MDA abstraction layers [20], and following the procedure to construct a conceptual framework to guide IS research [54], a metadesign (see Table 2) was formulated to guide the development of pimUML. It contains relevant concepts from enterprise ontology [23] and conceptual schema-centric development [55], positioned in both the MDA layers as well as in the layered enterprise software architecture layers [56]. In each column, the semantics captured by the listed constructs add to the semantics captured by constructs listed in the column(s) to the left. In other words, each abstraction level should include the constructs listed in each relevant architectural layer in addition to those listed at higher abstraction levels, thus reducing the abstraction through *semantic enrichment*. The metadesign aided in scoping pimUML by specifying which architectural layers should be excluded to ensure platform independence. For example, while a domain model should be included at the PIM level, a database schema should not, as this could make the PIM dependent on specific database solutions. Moreover, everything related to presentation should be excluded from the PIM, as UI design can vary greatly depending on the target platform.

4.2. pimUML

As Standard UML, xUML, and fUML all belong to the UML family, various notions from these three profiles were combined to define a novel UML profile called pimUML to define a (business) semantically

[3] http://www.mendix.com

[4] The complete Mendix metamodel is described in https://docs.mendix.com/apidocs-mxsdk/mxsdk/mendix-metamodel/

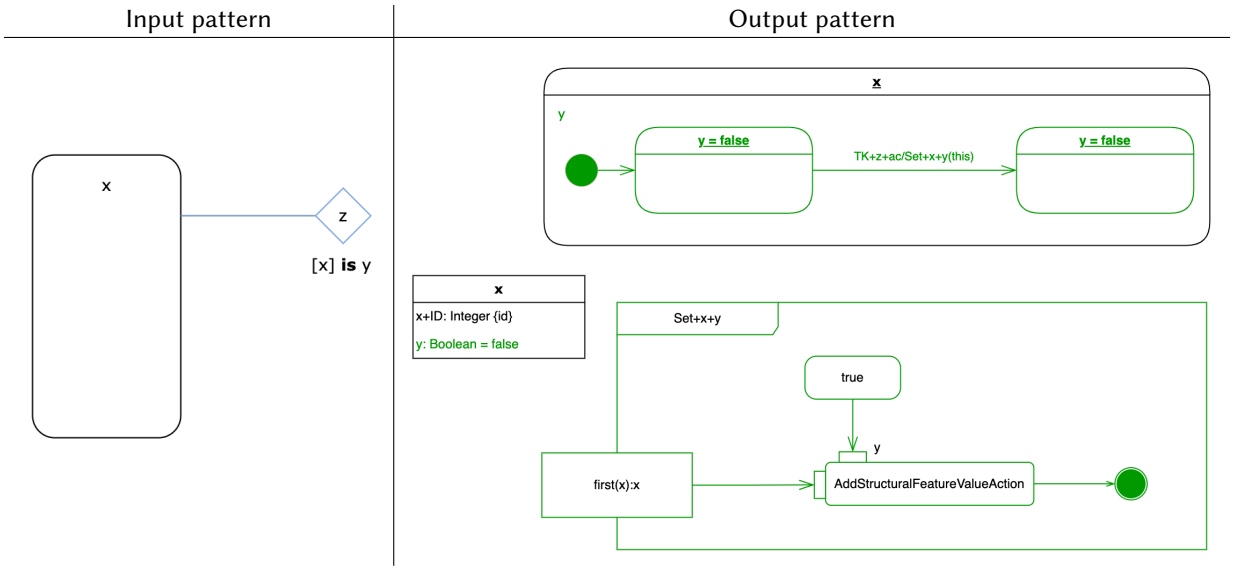
Table 2
Metadesign for the MDA abstraction layers: relevant concepts and decisions for enterprise applications in terms of the four-layered architecture

	CIM	PIM	PSM
Presentation			Views Layout Navigation UI elements Styles
Business	Vocabulary Business entities and relationships Business rules Business events	Data algorithms Data model Inference mechanism	Data representation Data structures Message formats Control structures
Persistence		CRUD operations Data integrity constraints	CRUD commands Data mapping State management Naming conventions
Source			Physical data DBMS structure DB schema design

rich PIM. The pimUML metamodel [20] is a subset of the latest version of UML (version 2.5.1), including the Action Language for Foundational UML (Alf) and the Object Constraint Language (OCL). Constructs included prioritize platform independence over diagram readability, and capture either structural, behavioral, or execution semantics:

- For *structural semantics*, the UML class diagram is used to capture business entities, as well as their attributes and relationships (data model).
- For *behavioral semantics*, UML state machines are used to model the life cycles of entity instances (inference mechanisms). Transition triggers express what transition occurs in response to what event, transition effect behaviors express what operation must be executed to realize that state transition, and the state entry events express what post-operations must be executed in response to the state transition.
- For *execution semantics*, the UML activity diagram is used to model the (CRUD) operation related to the entities. Actions included in pimUML are exclusively those predefined in the official UML

Figure 4: Example CIM-to-PIM mapping rule



specification in [46]. Activities are supplied an object as a parameter. Control flow edges denote the step-by-step ordering of action execution, while object flow edges denote the flow of data objects between actions. Additionally, Alf can be used to specify precise execution semantics when necessary.

4.3. Transformation mappings

While defining pimUML, the transformation mappings, in terms of mapping rules (see Fig. 4 for an example) and helper functions, from the DEMO Fact Model to pimUML and from pimUML to Mendix where updated constantly to ensure practical applicability. As the transformation mappings are between modeling languages of different families, the resulting transformations are *exogenous, out-place transformations* [57]. The mappings from the DEMO Fact Model to pimUML and from pimUML to Mendix are shown in Tables 3 and 4, respectively. The execution semantics in the PIM are inferred in such a way that they ensure that the business semantics are realized in a running system, primarily by specifying CRUD operations. For example, it makes sure that a derived entity instantiation receives an identical identifier value from the entity from which it was derived upon its creation.

Table 3
Mapping specification from the DEMO Fact Model to pimUML

ID	DEMO FM	pimUML
A1	Value type - user-declared, categorical	Enumeration (class diagram)
A2	Value type - user-declared, non-categorical	User-defined data type (class diagram)
A3	Declared entity type	Class (class diagram); Logical identifier (integer) (class diagram); State machine (state machine diagram)
A4	Derived entity type	Class (class diagram); State machine (state machine diagram)
A5	Attribute type	Property (class diagram)
A6	Event type	Boolean attribute, default false (class diagram); Orthogonal region for corresponding Boolean (state machine diagram); Initial node (state machine diagram); False state and true state (for corresponding Boolean) (state machine diagram); Transition from initial node to false state [t1] (state machine diagram); Transition from false state to true state [t2] (state machine diagram); Trigger on t2 with Alf expression for transition effect behavior activity (state machine diagram); Set<entity><event> activity (activity diagram)
A7	Specialization of entity type with *event type*	Logical identifier for child class (class diagram) Directed association from child to parent with "parent" role (state machine diagram); Entry activity in corresponding true state of the parent (state machine diagram); Create<child> activity (activity diagram)
A8	Specialization of value type with *event type*; Property type	Association class (class diagram)
A9	Specialization of entity type with *derivation rule*	Identifier property for child class (class diagram)l Directed association from child to parent with "parent" role (class diagram); OCL expression for derivation rule (class diagram; Create<child> activity to create new instance of child class (activity diagram)
A10	Generalization	Abstract parent (class diagram); Generalization (class diagram)
A11	Aggregation	Identifier property in whole class corresponding to each part class (class diagram); Association with shared aggregation adornment (class diagram)
A12	Property type; Cardinality laws	Association (class diagram); Multiplicities (class diagram)

4.4. Demonstration and evaluation

A paramount requirement of the PIM is that it must adequately capture the business semantics expressed in the CIM. To evaluate this, the EU-Rent case [22] was used, of which the DEMO model is presented in [23] (Rent-A-Car). After performing the necessary transformations, both the resulting pimUML and Mendix models were assessed to determine whether each fact type in the CIM (DEMO) was preserved

Table 4
Mapping specification from pimUML to Mendix

ID	pimUML	Mendix
B1	Class	Entity (domain model)
B2	Abstract Class	Entity (domain model)
B3	Enumeration; EnumerationLiteral	EnumerationAttributeType (domain model); Enumeration (domain model)
B4	User-defined data type	Entity (domain model)
B5	Association Class	Entity (domain model); Associations [domain and range] (domain model)
B6	Property (`isID == true`)	Attribute Integer (domain model)
B7	Property (of Datatype)	Attribute (domain model)
B8	Property (of Class)	Attribute (domain model)
B9	Association	Association (domain model)
B10	Generalization	Generalization/Specialization (domain model)
B11	State Machine	ACR Microflow (domain model + microflow)
B12	Region; Pseudostate (initial)	Call Workflow activity (microflow); Workflow (workflow); Start element (workflow)
B13	Activity	Microflow (microflow)
B14	InitialNode	StartEvent (microflow)
B15	ActivityFinalNode	EndEvent (microflow)
B16	Parameter; ActivityParameterNode	Entity reference (microflow); Parameter (microflow)
B17	ValueSpecificationAction; AddStructuralFeatureValueAction	Change Object (microflow)
B18	AddStructuralFeatureValueAction	ChangeObjectAction (microflow)
B19	CreateLinkAction	ChangeObjectAction (microflow)
B20	CreateObjectAction	CreateObjectAction (microflow)
B21	ReadStructuralFeatureAction	CreateVariableAction (microflow)
B22	Control flow	Sequence flow (microflow)
B23	Transition; Trigger	CallMicroflowTask (workflow); WaitForNotificationActivity (workflow)
B24	State entry activity	CallMicroflowTask (workflow)

in the PIM (pimUML) and the PSM (Mendix). The results are shown in Table 5. The only fact statement that was deemed to have not been sufficiently preserved is *aggregate entity type car group * year exists* at the PSM level. This is due to the fact that Mendix does not have specific construct in its metamodel to denote entities which are composed of other entities, thus constituting an aggregate entity. Instead, the aggregate entity type {CAR GROUP} * {YEAR} can be traced from the EU-Rent DEMO fact model to being implemented as a regular entity (with associations) in Mendix.

To compare results with the original (direct) mapping from the DEMO Fact Model to Mendix, the same analysis is performed after executing the transformations as described in [6] for the same case.

Table 5
Semantic preservation, expressed as the nr of facts, of the approach through pimUML compared to the original direct transformation [6] from the DEMO Fact Model to Mendix for the EU-Rent case.

		through pimUML		direct
Concept from DEMO Fact Model	*CIM*	*PIM* (pimUML)	*PSM (Mendix)*	*Mendix*
Entity Types	12	12	11	7
Value Types	3	3	3	3
Event Types (incl. concern link)	14	14	14	14
Attribute Types (incl. domain and range)	42	42	42	36
Property Types (incl. domain and range)	42	42	42	33

The results (see right most column of Table 5) show that an improvement was made, mainly because the original approach was not able to deal with the specializations that were present in the EU-Rent case.

5. Conclusion

This research aimed to improve the extensibility of the existing DEMO to Mendix transformation to other IT-platforms while, "opening up" for other (complementary) CIMs besides DEMO models. Following the MDA approach, pimUML was developed as a neutral intermediate language in the transformation from DEMO to Mendix. This newly defined UML profile captures both business semantics and adds execution semantics, while it remains (IT) platform independent. The demonstration and evaluation showed that the semantic loss was very low and that pimUML is capable of retaining a high degree of business semantics, while it is aimed at software design – as opposed to the CIM level that is focused on capturing business semantics, requirements, and processes. As such, pimUML serves multiple purposes: code generation, stakeholder communication, reducing technology lock-in, and increasing enterprise agility. We end our conclusions with a discussion on the potential use of (generative) AI and some topics for future research.

The rise of generative AI (GenAI) technologies, AI-assisted software development, and AI agents, enables new ways to quickly create enterprise applications, and may put research into model-driven software engineering in the shade. However, such AI-based approaches have to be considered non-deterministic (closed) boxes [58], that are not able (yet) to provide the rationale for design choices [59, 60]. As pimUML uses a restrictive subset of UML, limiting the design choices, pimUML has the potential to serve as a means to add a degree of explainability and determinism, allowing software engineers to understand what is "under the hood" of software designs generated by AI technologies.

We see opportunities in using AI to quickly generate enterprise models from enterprise (process) data, or to improve upon those models. Only by adopting known and mathematically sound transformation mappings, one can ensure repeatability and transparency in generating enterprise software from higher-level (enterprise) models.

Finally, we see five key directions for further research:

Extending pimUML – By including the other DEMO aspect models at the CIM level, pimUML could be enhanced. For example, the DEMO Process Model and Action Model can be used to specify the business processes and rules respectively. Moreover, additional implementation choices [61] can be defined and included, such as whether a task is performed by a human or by a piece of software. This may require pimUML to capture additional constructs and/or views.

Targeting different platforms – To account for complexity induced by variety on both the business and its supporting IT (platforms), a PIM should be able to handle as large a variety of input, and produce as large a variety of desired output as possible. To truly test the (IT) platform independence of pimUML, the ability of pimUML to be mapped to different IT platforms or technologies – ranging from high code to low code to no code – should be further explored.

Discerning different levels of (IT) platform independence – There is likely to be a need for more nuance regarding IT platform independence. Consider, for example, the development of different human-computer interfacing technologies. The "modality" of the interface, be it, e.g., a traditional screen or forms based interface, a voice based interface, or a virtual reality based interface, will have a profound impact on the way business processes can be supported using IT. As a consequence, creating a PIM for a given CIM, may need a decision regarding the "modality" of the computer interfacing to be used, de facto reducing the platform independence of the resulting PIM. This certainly requires further investigation, to clarify different dimensions and levels of platform independence.

Targeting different enterprise modeling languages – As with the aim to "open the door" to other or complementary CIMs, it is wise to look at other enterprise modeling languages, such as 4EM, BPMN, EAML, MEMO, SBVR, and SysML. Such modeling languages could either be used in addition or as an

alternative to DEMO where needed, depending on, e.g., its expressiveness or familiarity to a given (modeling) community.

Automating the transformations – While the transformation mappings designed in this study were demonstrated and evaluated manually, automating these transformations would facilitate a more effective evaluation of pimUML. Moreover, once the transformations to and from pimUML are automated, tested, and refined accordingly, tool support can be developed, allowing pimUML to be used in real-world projects to assist enterprises in improving business-IT alignment.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] D. L. Olson, S. Kesharwani, Enterprise information system trends, in: J. Filipe, J. Cordeiro (Eds.), Enterprise Information Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 3–14.

[2] S. Mithas, F. W. McFarlan, What is digital intelligence?, IT Professional 19 (2017) 3–6.

[3] R. Dove, Agile enterprise cornerstones: Knowledge, values, and response ability, in: R. L. Baskerville, L. Mathiassen, J. Pries-Heje, J. I. DeGross (Eds.), Business Agility and Information Technology Diffusion, Springer US, Boston, MA, 2005, pp. 313–330.

[4] Gartner, Taming the Digital Dragon: The 2014 CIO Agenda, Technical Report, Gartner Executive Programs, 2014. URL: https://www.gartner.com/imagesrv/cio/pdf/cio_agenda_insights2014.pdf.

[5] K. Hinkelmann, A. Gerber, D. Karagiannis, B. Thoenssen, A. van der Merwe, R. Woitsch, A new paradigm for the continuous alignment of business and IT: Combining enterprise architecture modelling and enterprise ontology, Computers in Industry 79 (2016) 77–86.

[6] M. R. Krouwel, M. Op 't Land, H. A. Proper, From Enterprise Models to Low-Code Applications: Mapping DEMO to Mendix, illustrated in the Social Housing domain, International Journal on Software and Systems Modeling 23 (2024) 837–864.

[7] Z. Hemel, L. Kats, D. Groenewegen, E. Visser, Code generation by model transformation: A case study in transformation modularity, SoSyM 9 (2009) 375–402.

[8] A. W. Wymore, Model-Based Systems Engineering, first ed., CRC Press, 1993.

[9] S. Beydeda, M. Book, V. Gruhn (Eds.), Model-Driven Software Development, Springer Berlin, Heidelberg, 2005.

[10] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, K. Czarnecki, B. von Stockfleth, Model-Driven Software Development: Technology, Engineering, Management, Wiley Software Patterns Series, Wiley, 2013.

[11] D. Di Ruscio, D. Kolovos, J. Lara, A. Pierantonio, M. Tisi, M. Wimmer, Low-code development and model-driven engineering: Two sides of the same coin?, SoSyM 21 (2022) 437–446.

[12] Object Management Group, MDA guide rev. 2.0, 2014. URL: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01, last visited 5-Jul-2025.

[13] P. Barbosa, F. Ramalho, J. Figueiredo, A. J´unior, A. Costa, L. Gomes, Checking semantics equivalence of mda transformations in concurrent systems, Journal of Universal Computer Science 15 (2009) 2196–2224.

[14] M. Kardos, M. Drozdova, Analytical method of CIM to PIM transformation in model driven architecture (MDA), Journal of Information and Organizational Sciences 34 (2010).

[15] M. R. Krouwel, M. Op 't Land, H. A. Proper, Generating Low-Code Applications from Enterprise Ontology, in: B. S. Barn, K. Sandkuhl (Eds.), PoEM 2022: The Practice of Enterprise Modeling, volume 456 of *Lecture Notes in Business Information Processing*, Springer Nature Switzerland AG, 2022, pp. 19–32.

[16] I. Alfonso, A. Conrardy, J. Cabot, Towards the interoperability of low-code platforms, in: L. Pu-

fahl, K. Rosenthal, S. España, S. Nurcan (Eds.), Intelligent Information Systems, Springer Nature Switzerland, Cham, 2025, pp. 3–11.
[17] I. Alfonso, A. Conrardy, A. Sulejmani, A. Nirumand, F. Ul Haq, M. Gomez-Vazquez, J.-S. Sottet, J. Cabot, Building BESSER: An open-source low-code platform, in: H. van der Aa, D. Bork, R. Schmidt, A. Sturm (Eds.), Enterprise, Business-Process and Information Systems Modeling, Springer Nature Switzerland, Cham, 2024, pp. 203–212.
[18] V. Freitas, D. Pinto, V. Caires, L. Tadeu, D. Aveiro, The DISME low-code platform - from simple diagram creation to system execution, Proceedings of the 22nd CIAO! DC and EEWC (2022). URL: https://ceur-ws.org/Vol-3388/.
[19] K. Conboy, R. Gleasure, E. Cullina, Agile design science research, in: B. Donnellan, M. Helfert, J. Kenneally, D. VanderMeer, M. Rothenberger, R. Winter (Eds.), New Horizons in Design Science: Broadening the Research Agenda, Springer International Publishing, 2015, pp. 168–180.
[20] N. A. Bzowski, A Model Driven Architecture Transformation from Ontological Enterprise Models to Low-Code, Diploma thesis, Technische Universität Wien, 2025. ReposiTUm.
[21] E. Domínguez, M. A. Zapata, Mappings and interoperability: A meta-modelling approach, in: Proceedings of the First International Conference on Advances in Information Systems, ADVIS '00, Springer-Verlag, Berlin, Heidelberg, 2000, p. 352–362.
[22] M. Schacher, Mini EU-Rent: Business Model, Technical Report, KnowGravity, 2008. URL: http://www.knowgravity.com/pdf-e/Mini%20EU-Rent%20BU.pdf.
[23] J. L. G. Dietz, J. B. F. Mulder, Enterprise Ontology — A Human-Centric Approach to Understanding the Essence of Organisation, The Enterprise Engineering Series, Springer, Cham, 2024.
[24] K. Lano, S. Kolahdouz Rahimi, I. Poernomo, Comparative evaluation of model transformation specification approaches, Int J Software Informatics 6 (2012) 233–269.
[25] J. Krogstie, A. Sølvberg, Information Systems Engineering: Conceptual Modeling in a quality perspective, The Norwegian University of Science and Technology, 2001.
[26] M. R. Krouwel, On the Design of Enterprise Ontology-Driven Software Development, Ph.D. thesis, Maastricht University, 2023.
[27] Object Management Group, Model Driven Architecture (MDA) Guide, Technical Report, Object Management Group, 2014. URL: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01, rev. 2.0.
[28] N. M. J. Basha, S. A. Moiz, M. Rizwanullah, Model Based Software Development: Issues & Challenges, International Journal of Computer Science and Informatics 3 (2013).
[29] C. Heitmeyer, S. Shukla, M. Archer, E. Leonard, On Model-Based Software Development, in: J. Münch, K. Schmid (Eds.), Perspectives on the Future of Software Engineering, Springer, 2013, pp. 49–60.
[30] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Synthesis Lectures on Software Engineering, second ed., Morgan & Claypool Publishers, 2017.
[31] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, Y. Shaham-Gafni, Model traceability, IBM Systems Journal 45 (2006) 515–526.
[32] I. Santiago, Á. Jiménez, J. M. Vara, V. de Castro, V. A. Bollati, E. Marcos, Model-Driven Engineering as a new landscape for traceability management: A systematic literature review, Information and Software Technology 54 (2012) 1340–1356.
[33] M. Argañaraz, A. Funes, A. Dasso, An MDA approach to business process model transformations, SADIO Electronic Journal of Informatics andOperations Research 9 (2010) 24–48.
[34] V. De Castro, E. Marcos, J. M. Vara, Applying CIM-to-PIM model transformations for the service-oriented development of information systems, Inf. Softw. Technol. 53 (2011) 87–105.
[35] A. Bozzon, M. Brambilla, P. Fraternali, Conceptual modeling of multimedia search applications using rich process models, in: Proceedings of ICWE '9, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 315–329.
[36] J. L. G. Dietz, J. A. P. Hoogervorst, A. Albani, D. Aveiro, E. Babkin, J. Barjis, A. Caetano, P. Huysmans, J. Iijima, S. van Kervel, H. Mulder, M. Op 't Land, H. A. Proper, J. Sanz, L. Terlouw, J. Tribolet, J. Verelst, R. Winter, The discipline of enterprise engineering, International Journal of Organisational Design and Engineering 3 (2013) 86–114.

[37] R. E. Giachetti, Design of Enterprise Systems: Theory, Architecture, and Methods, first ed., CRC Press, 2010.
[38] J. L. Austin, How to do things with words, William James Lectures, Oxford University Press, 1962.
[39] J. Habermas, The theory of communicative action, Cambridge: Polity Press, 1986.
[40] J. R. Searle, Speech Acts: An Essay in the Philosophy of Language, Cambridge University Press, Cambridge, London, 1969.
[41] H. Weigand, Two decades of language/action perspective, NLE 49 (2006) 45–46.
[42] V. E. van Reijswoud, J. B. F. Mulder, J. L. G. Dietz, Communicative Action Based Business Process and Information Modelling with DEMO, The Information Systems Journal 9 (1999) 117–138.
[43] J. L. G. Dietz, DEMO Specification Language 4.9.1, 2024. URL: https://ee-institute.org/download/demo-specification-language-4-9-1/.
[44] Object Management Group, MDA specifications, 2025. URL: https://www.omg.org/mda/specs.htm, last visited 5-Jul-2025.
[45] H. Störrle, How are conceptual models used in industrial software development? A descriptive survey, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE '17, ACM, New York, USA, 2017, pp. 160––169.
[46] Object Management Group, Unified Modeling Language: Version 2.5.1, 2017. URL: https://www.omg.org/spec/UML/2.5.1/, last visited 5-Jul-2025.
[47] S. J. Mellor, M. Balcer, Executable UML: A Foundation for Model-Driven Architecture, Addison-Wesley, 2002.
[48] Object Management Group, Semantics of a foundational subset for executable UML models, 2021. URL: https://www.omg.org/spec/FUML/1.5, last visited 5-Jul-2024.
[49] A. C. Bock, U. Frank, Low-code platform, Business & Information Systems Engineering 63 (2021) 733–740.
[50] Y. Luo, P. Liang, C. Wang, M. Shahin, J. Zhan, Characteristics and challenges of low-code development: The practitioners' perspective, in: Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, New York, USA, 2021, pp. 1–11.
[51] IBM, What is low-code?, 2024. URL: https://www.ibm.com/topics/low-code.
[52] J. Sijtstra, Quantifying the effectiveness of low-code development platforms in the Dutch public sector, Master's thesis, LIACS, Leiden University, 2022. URL: https://theses.liacs.nl/2221.
[53] K. Rokis, M. Kirikova, Challenges of low-code/no-code software development: A literature review, in: Ē. Nazaruka, K. Sandkuhl, U. Seigerroth (Eds.), Perspectives in Business Informatics Research, Springer International Publishing, Cham, 2022, pp. 3–17.
[54] J. E. Chukwuere, Theoretical and conceptual framework: A critical part of information systems research process and writing, Review of International Geographical Education 11 (2021) 2678–2683.
[55] A. Olivé, Conceptual schema-centric development: A grand challenge for information systems research, in: O. Pastor, J. Falcão e Cunha (Eds.), Advanced Information Systems Engineering, volume 3520, Springer Berlin Heidelberg, 2005, pp. 1–15.
[56] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co., Inc., USA, 2002.
[57] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, Synthesis lectures on software engineering, second ed., Morgan & Claypool Publishers, 2017.
[58] S. Ouyang, J. M. Zhang, M. Harman, M. Wang, An empirical study of the non-determinism of ChatGPT in code generation, ACM Trans. Softw. Eng. Methodol. 34 (2025).
[59] Y. Liu, C. Tantithamthavorn, Y. Liu, L. Li, On the reliability and explainability of language models for program generation, ACM Trans. Softw. Eng. Methodol. 33 (2024).
[60] J. Sun, Q. V. Liao, M. Muller, M. Agarwal, S. Houde, K. Talamadupula, J. D. Weisz, Investigating explainability of generative AI for code through scenario-based design, in: Proceedings of the 27th International Conference on Intelligent User Interfaces, ACM, New York, USA, 2022, p. 212–228.
[61] M. R. Krouwel, M. Op 't Land, T. Offerman, Formalizing Organization Implementation, in: D. Aveiro, R. Pergl, D. Gouveia (Eds.), EEWC 2016: Advances in Enterprise Engineering X, volume 252 of *LNBIP*, Springer, Funchal, Madeira Island, Portugal, 2016, pp. 3–18.