Simplified Enterprise Modelling Platform Architecture

Mark A. T. Mulder[1], Henderik A. Proper[2,3], Fiodor Bodnar and Rick Mulder

[1]TEEC2, Hoevelaken, the Netherlands
[2]Luxembourg Institute of Science and Technology (LIST), Belval, Luxembourg
[3]University of Luxembourg, Luxembourg

Abstract

In the context of enterprise engineering and architecting, there is a need to capture many different aspects and perspectives of an enterprise in terms of models. The resulting enterprise models are typically expressed in different modelling languages. At the same time, since these models pertain to the same enterprise, it is desirable to ensure that they (potentially) form a coherent whole.

Capturing such collections of enterprise models involving different modelling languages while ensuring their coherence is a major challenge for modelling tools. Practice in the last years shows that the demand for collaborative modelling with flexible notations that could run online and across different platforms has risen. They also wanted to be able to use it in an international language setting, capturing all needed perspectives of the business. In this paper, we report on the architecture of a novel enterprise modelling platform which endeavours to meet these challenges.

This architecture involves a multi-meta model approach, supporting different notations, models, and the inclusion of actual sample instances to enable validation. The resulting modelling platform supports concepts such as multi-perspective modelling, verification by instantiation and narrative analysis.

Keywords

enterprise engineering, enterprise architecture, modelling tools,

1. Introduction

This paper pertains to the development of enterprise modelling tool support from a practice-oriented perspective. We will discuss this tool as a platform to emphasise the broader context of the usage of this modelling environment. The platform is called Simplified and will be available online for academic and commercial use [1].

In the context of enterprise engineering and architecting, there is a need to capture many different aspects and perspectives of an enterprise in terms of models. The resulting enterprise models are typically expressed in different modelling languages. At the same time, since these models pertain to the same enterprise, it is desirable to ensure that they (potentially) form a coherent whole [1, 2]. A start has been made on scientifically approaching a multi-modelling environment [3, 4] but was not finished. Other initiatives that support multiple modelling

PoEM 2022 Forum, 15th IFIP Working Conference on the Practice of Enterprise Modeling 2022 (PoEM-Forum 2022), November 23-25, 2022, London, UK
markmulder@teec2.nl (M. A. T. Mulder); e.proper@acm.org (H. A. Proper)
0000-0002-1846-0238 (M. A. T. Mulder); 0000-0002-7318-2496 (H. A. Proper); 0000-0001-6266-0985 (F. Bodnar); 0000-0002-9500-0702 (R. Mulder)

[1]https://simplified.engineering

languages are the ADOXX [2] platform, Metacase [3] and, to a certain degree, the Sparx Enterprise Architect (SEA)[4].

Capturing such collections of enterprise models involving different modelling languages while ensuring their coherence is a major challenge for modelling tools. We will address several challenges in the meta-level of structuring modelling languages, which we call notations to avoid confusion with natural languages. The model level has challenges in ownership of elements, connections, and the semantic value of those elements and connections. On the instance level, storage and connectivity pose a challenge. After having addressed these, the presentation and visualisation will require attention.

In this paper, we report on the architecture of a novel enterprise modelling platform which endeavours to meet these challenges. This architecture involves a multi-meta model approach, supporting different notations, models, and the inclusion of actual sample instances to enable validation. The resulting modelling platform supports concepts such as multi-perspective modelling, verification by instantiation, and narrative analysis.

As discussed in [5], the way toward the Simplified modelling platform started with the PhD research project on the automatic verification and exchange of DEMO models [6]. The Design and Engineering Methodology for Organisations (DEMO) [7, 8] method is a core method (based on a theoretically founded method*ology*) within the discipline of Enterprise Engineering (EE) [9]. The DEMO methodology focuses on creating so-called *essential* models of enterprises.

As part of this PhD work, requirements for tooling were defined to find the most suitable tool for DEMO modelling, where a broader comparison of available tooling was made [6]. Among other requirements inspired by practice in modelling, the tool would have to support at least the essential model of an organisation [8]. The lack of good tooling for DEMO modelling prompted us to start the development of a new tool. This initiative resulted in the creation of *Plena*, an add-on to an existing SEA modelling tool. Plena supports the linking of narrative text by the import of the text lines of an interview along with the modelling of all aspect models of the essential model. We have been modelling DEMO in combination with ArchiMate in this enterprise-grade tool [10] that was able to capture the complexity of modelling an enterprise with multiple modelling languages. During the further development of Plena, it became apparent that the underlying SEA modelling software was increasingly becoming a limiting factor which impeded the implementation of the desired features.

Next to SEA, ADOXX does not seem to support multiple models simultaneously and, additionally, it is not cloud-based, and as such, it is not suitable for our goals. Metacase is limited to a modelling environment and does not contain the tools for presenting and reporting the created models in the way we formulated our requirements. Practice in the last years shows that the demand for collaborative modelling with flexible notations that run online cross-platform (e.g. iOS, Windows, Linux) to be used in an international language setting, capturing all needed perspectives of the business has risen. The need for translating connectable notations with different semantics [11, 12, 13, 14] required a flexible but integrated way of model storage. While modelling, educational coaching [15] is also an aspect that did not find enough support in existing

[2] https://www.adoxx.org/

[3] https://www.metacase.com/

[4] https://sparxsystems.com/

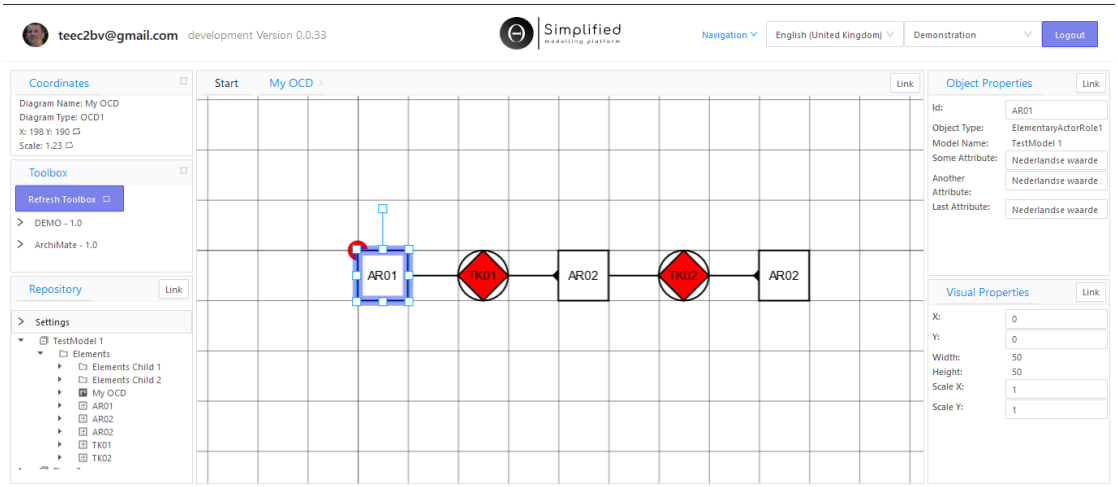

Figure 1: An overview of the modeller in the simplified platform

tools. In the end, we found no suitable tooling to support all the requirements.

Though the back-end of SEA was quite capable of storing the models, the user interface started lacking performance and comprehensibility. The performance aspect became an issue as communicating with the back-end through the available UI-API was not optimised for more significant amounts of generation and interaction. The comprehensibility suffered from a lack of available customisation. This was because add-ins in SEA have only one panel in the UI, which forced us to put increasing amounts of functions in the same amount of visual space. The number of options in this panel was too much to be easily understood.

Considering the limitations of the SEA software and other tools, we have decided to develop a standalone web-based modelling platform, which would allow us to implement all the required features.

This development led to a decision to stop the old solution's development and start from scratch to build a reliable, enterprise-grade, scalable solution that could capture metamodels, models and instances. At the same time, this platform must be able to help others avoid having to reinvent the wheel for every modelling project they start.

Due to this change in philosophy, we decided to add the requirements to let the platform be a low barrier modelling platform where people who are new to modelling can start, make this platform have multi notation modelling support where the user has an integrated business and technology overview and can experience integrable modelling and usage of those models.

The architecture described in this paper supports concepts like multi-perspective modelling, verification by instantiation, and narrative analysis. Though not every aspect is implemented now, we see the practical need. We will be actively expanding the functional scope to create a collaboration platform from the perspective of both the modeller and the organisations enabling this modelling.

Like with the previous tool, we are inspired from a DEMO modelling perspective and have expanded the platform with ArchiMate, BPMN, VISI[5] and Sticky Notes (a.k.a. Post-it) as test

[5]https://www.bimloket.nl/p/557/VISI

cases. The domain-specific variant of DEMO from an implementation perspective, 'Voorwaarden scheppen voor de Invoering van Standaardisatie ICT in de bouw' (VISI), is added to the equation to perform our verification by instantiation on building the architecture. Other notations like e3Value, IDEF0/3, EPC, Petri Net, and common shapes are next on our development list when all test cases have been approved.

The structure of the remainder of this paper is as follows. Section 2 provides more background to the architecture of the Simplified platform. In section 3, we discuss the metamodel of the platform storage as a modelling platform should support it. Section 4 then continues on the specific layer of storing notations. section 5 reports on the model layer of the storage. Finally, before concluding, section 6 looks ahead to the instance layer of the storage we are moving to.

2. Architecture

The Simplified platform consists of six functional layers, divided into two servers: the application server (containing the interface, message, process, cache, and persistence layers) and the database server (database layer) as depicted in fig. 2.

First, the interface layer consists of two interfaces for accessing the platform. The REST/JSON API is the most straightforward interface, allowing the communication of data that requires no authentication. This information includes the information on installed public notations (e.g. ArchiMate, DEMO, BPMN) and the creation of free accounts. No modelling information can be exchanged using this API. This API exists because some information needs arise at a point when clients can provide no authentication yet. Therefore, this information is public and can be retrieved by anyone or anything calling this API.

The other interface that facilitates access to the platform is an authenticated asynchronous web socket messaging interface. The authorised developers can also use this interface to build their interactions. It is designed to receive and handle universal messages for both requests and responses, and the structure is the only fixed information. The messages contain a request type and a payload whose structure, in turn, is related to the request type. The returned message contains a result of successful processing or an error in case the request is not processed correctly. After

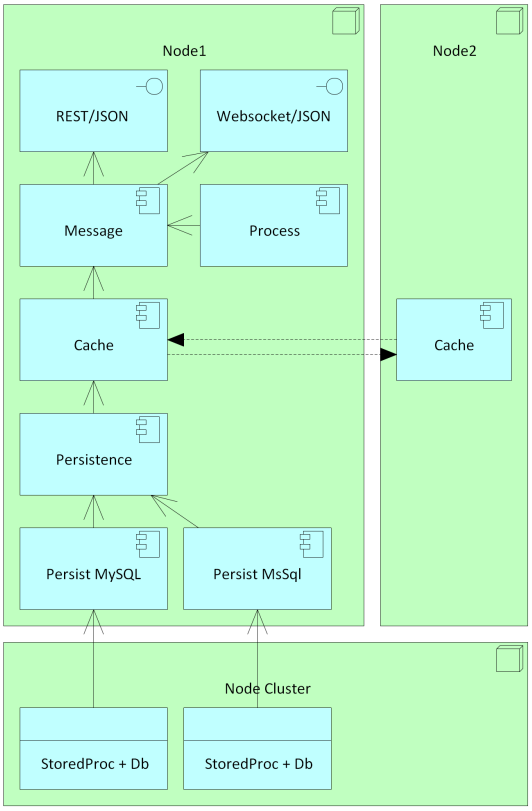

Figure 2: Architecture Layers

successful message processing, the interface will broadcast the result to all relevant connected users. It is worth noting that the communication is asynchronous, meaning you do not know when the server will answer the request. In most cases, it does not matter as long as it is answered. In other cases, the client can provide a message Id to keep track of the request and corresponding response flow. Beware that the client can receive information he did not request due to the broadcast system.

Next, the message layer handles messages. In this layer, the message is broken down into internal structures to ensure that no illegal content is included and that the content is not sent to the wrong functionality. The message layer is also used to combine primary functionality, e.g. retrieving different entities for one request, to create more efficient requests. The permission check for functionality is also implemented in the message layer and checks the availability of the requested functionality for the specific user. Complex requests are handled in the process layer, e.g. auto-layout, extended functionality, and import and export. We will not discuss this part of the functionality in this paper since the implementation and application of it is a paper on its own. Simple requests, e.g. model element retrieval, are directly forwarded to the cache layer.

The cache layer contributes to the fast retrieval of information and the load distribution between multiple servers. Cache servers announce themselves to a central (database) storage administration and provide a part of the cache of the complete data set. A challenge, and the subject for future research, in the cache layer, is the dependency handling of individual data entries. Dependency handling has been described in many forms but often results in inefficient storage or complex dependencies that reduce the efficiency, making the cache obsolete. The parts of the cache layer that have this issue have been bypassed for now. Another function built into the cache layer is the permission or security information, e.g. read access or grant/deny options. The combination of security and permissions create a fine-grained access model that can be used to restrict users to specific models and functionality. Information retrieval of permissions needed to execute specific lower-level functionality can be implemented in either the message or the cache layer. When the permission check is put in the message layer, we get a mix of allowing access and checking for illegal information simultaneously. If the permission check is put in the cache layer, we get a mix of storage and access checks. We have chosen the cache layer to enforce access control to prevent data from being retrieved without permission. The cache layer will, in turn, send data or initially retrieve the information from the persistence layer.

Next, the default value functionality is administered in the persistence layer. One can argue whether this should be in another layer. The configurable persistence layer will activate the right persistence provider that triggers the correct driver to communicate information to the database layer. The persistence layer performs the last check to verify the content of yet-to-be-saved content and then chooses the storage medium, i.e. MySQL, MsSQL or memory. Though this layer is thin, creating a separation of concerns is essential.

Lastly, in theory, the database cluster realising a database layer can be any database. The implementation has reached the Microsoft SQL server, the open source MySQL server and memory storage mainly used for unit testing. The storage layer has the relational integrity functionality to ensure that the dependencies are present in the storage. Another function given to this layer is the retrieval of complex combinations of information. This is done through complex queries containing, for example, inner joins in the case of MsSQL and MySQL, and nested loops in the case of memory storage.

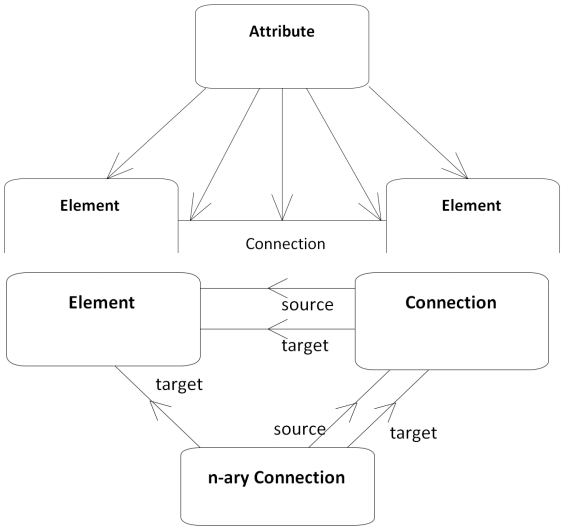

Figure 3: Attributes

Figure 4: n-ary connection storage

This architecture benefits the user as its design handles security and evolvability. From a usage perspective changes in the technologies of the platform do not interfere with the usage. The API design makes the usage simple and is a currently wide used standard. Furthermore, the authentication and messaging structure provide a transparent usage of the model information and prevent a platform lock-in. Next, the caching provides a performance gain for research that requires frequent model look-ups. Lastly, the user can benefit from the persistence flexibility by requesting adding storage to new facilities.

3. Meta^n-model

A model, in the base, consists of elements and connections. The elements and connections can have attributes often referred to as properties. The attributes are based on attribute types which can also be interpreted as data types during implementation. When describing a metamodel, we come to the same concepts of elements and connections with their respective attributes. Therefore, these three concepts form the base for both the model and the metamodel.

Figure 3 shows how attributes relate to the elements and connection. As said, elements and connections can have attributes, but the same holds for connection's begin and end positions. Therefore, every connection has three sets of attributes, and every element has a single set of attributes.

Binary relations between elements are more common in modelling but do not cover all modelling requirements. Consequently, the concept of connection has to be explained for binary relations but also needs to hold for n-ary relations. In implementation, in a relational database, an n-ary relation without a central concept modelled as an element is very complex. This is due to how an n-ary relation is conceptualised and that every link will become a record in a connection entity. Therefore, we have distinguished binary and n-ary relations in our metamodels. Figure 4 shows how the binary relations will be stored in a connection entity and the n-ary connections in a separate entity. N-ary connections can be between elements or connections; therefore, the source

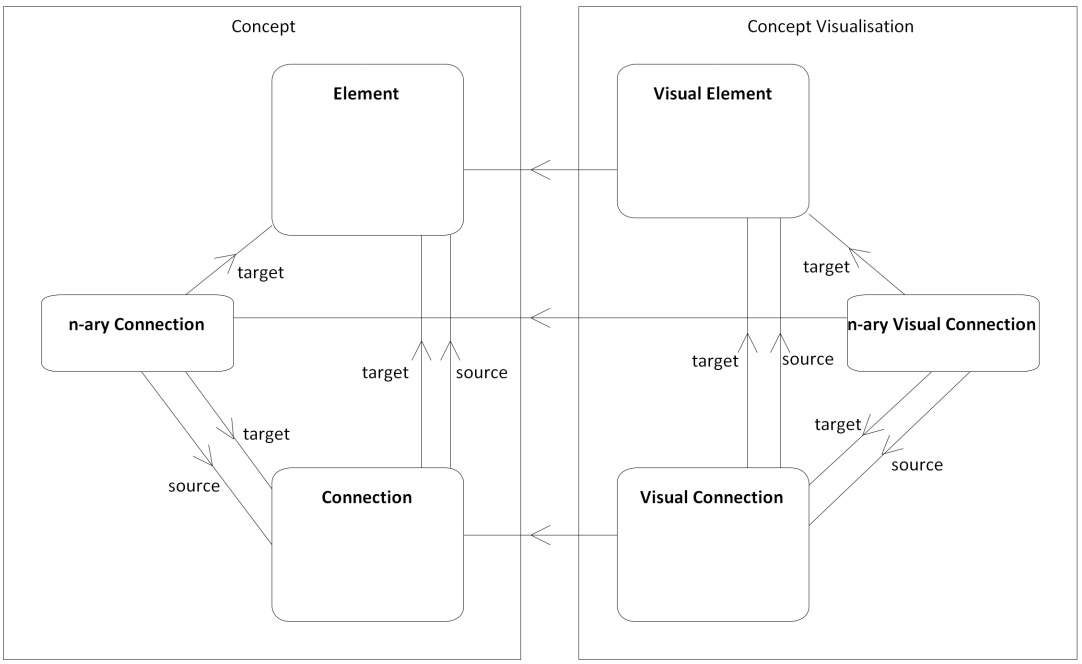

Figure 5: Concepts versus Visual Concepts

of an n-ary connection is always a connection, and the target can be either a connection or an element.

In fig. 5 the concepts are shown next to their visual counterparts. It is worth noting that the visual concepts mirror the concepts. Visual concepts reference their model counterpart and, therefore, must pre-exist. Concepts and visual concepts have a one-to-many relation. As such, multiple visual elements and visual connections can exist, i.e. on a diagram that references the same element or connection, respectively. It is possible to visualise only a selection of concepts. This is also true for n-ary connections, in that they can be displayed as a lower level n-ary connection or simply as a binary connection.

4. Notations

We define the notation level as the highest meta-level in the Simplified platform. At this level, we can define how models can be modelled and what rules the models must adhere to. We can formulate notations in two distinct ways. One of the methods has already been implemented and consists of a grammar describing the notation. We have briefly touched upon this topic [5]. The principle of the notation grammar lies in defining all elements, connections and rules in such a way that the automated processing of the script results in unambiguous handling of modelling possibilities. The other method is described at the end of this section.

The architecture uses dynamic metamodels that restrict the models on run-time. The notation architecture structure is visualised in fig. 6.

To be able to create a model and a visualisation of that model, one has to agree on a notation

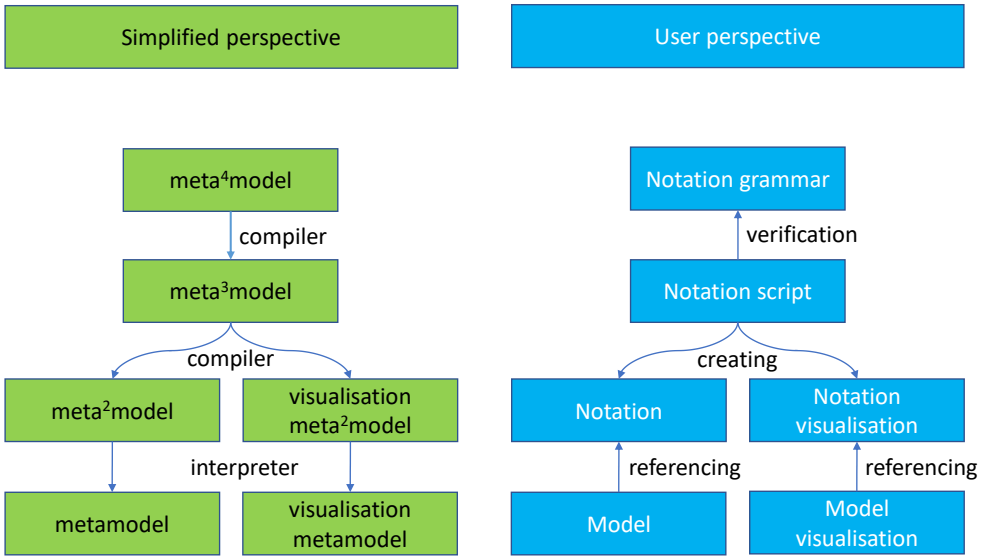

Figure 6: Notations from different perspectives

for both aspects. This results in creating a metamodel. In order to describe this metamodel, we have created a scripting language that can define notations. This scripting language is described in a grammar where the levels of abstraction are reflected in the platform, and the transformation between the levels is either interpretation or compilation.

The current version of the modelling script has a grammar for the following list of modelling concepts.

Element is the base concept often referred to as 'block'. An element can have attributes, or properties, that have types. From a mathematical concept, the element is a node or a vertex.

Connection is the base concept often referred to as 'arrow'. A connection can have attributes too. From a mathematical perspective, the connection is an edge or a link.

Typedef is the type to represent the attribute. The supported types are text, enumeration, Boolean, DateTime, integer, URL, and UUID. Within these types, restrictions on the types and the definition of the enum can be added.

Toolbox focuses on available elements and connections for a specific perspective. The perspective is chosen by the diagram, table, or cube that is presented in the UI.

Virtual Element is the concept of an element that starts to exist when a specific combination of elements and connections starts to exist. The virtual element does exist in the model but is mainly used for the visualisation of models.

Rule is the way to restrict the model. By defining a logic query on the model, one can generate messages showing rules that have been violated. The violation of a rule will never result in the rejection of a modelling action because partial models will most likely violate a rule.

Table is a visualisation of model information in a textual column-shaped format. Tables can be defined to list information in the scope of a repository, model, or a single diagram. This representation, like others, is updated in real-time with model changes.

Visual defines the shape of a model element or a model connection that can be displayed on a diagram.
Diagram is a special element that can hold other elements to visualise a specific perspective of the model.
Cube is the multi-dimensional textual representation of a part of a model. Currently, a two-dimensional cube, matrix, is the only supported representation.
Behaviour defines the actions in the UI needed to realise the modelling process described in the methodology accompanying the notation.

A notation can be self-contained, extending or replacing concepts from other notations. By extending other notations, one can add own elements without redefining previous notations. With the replacement functionality, one can restrict the use of a notation to a smaller subset of concepts or other attributes of those concepts.

We are in the process of creating the second method of notation definition by an elevator that can promote a model to a notation. This model will be an instance designed with a notation-notation. The functionality will be equal to the notation script functionality. With this functionality, the definition of a notation can be either textual, graphical, or a combination of both.

During our journey into the notation script, we have found new options in every notation [16] so far[6] (e.g. DEMO , BPMN, VISI, ArchiMate, IDEF0). New modelling concepts will be incorporated into grammar for the modelling script when needed. Future research will include the possibility of extending the structure of notations with the process of creating the notation in such a way that the platform supports a complete methodology and users are guided in the creation of models.

5. Models

In the model meta-level, we have the model concepts of an element, connection and attribute that have an internal structure, which are explained later in this section. The model itself is an unordered set of model elements which can be interrelated. In this model, we can have a model element that is the objectification of a projected concept in the world. Furthermore, a model can have a connection which is the objectification of the relation between two or more elements or connections. One can use the n-ary model connections for modelling multiple connected real-world concepts, e.g. 3-ary: renting a specific car at a particular location with a designated driver. Also, connections between connections are needed, e.g. mutually exclusive relations where one of two connections is valid at a specific time.

As shown in fig. 7, concepts (elements, connections, and attributes) are contained in models that are, in turn, held in repositories with no logical limit on the number of models in a repository. A model contains model folders which in turn can contain other model folders and elements and are used for a logistical overview of the model. Connections will not be stored in folders because they represent a relation and thus are essentially virtual, but for modelling purposes have been objectified. Not showing the connection in a folder is a design choice and both showing and not showing have been seen in various other tools. Elements can be specialised into multiple types, e.g. (basic) Element, Diagram, Virtual Element, Table. Diagrams contain visual concepts,

[6] https://gitlab.com/teec2/simplified/notations

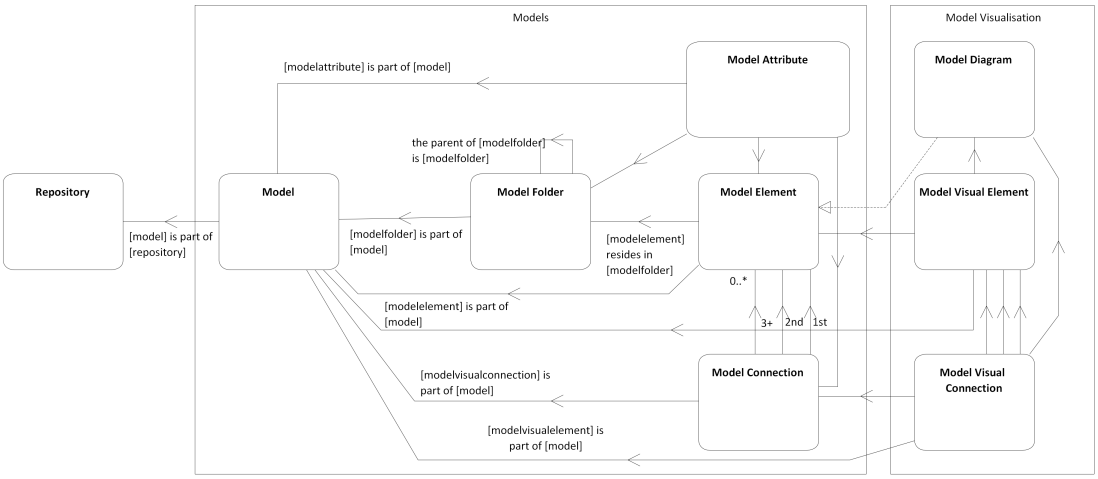

Figure 7: Model storage structure

and, as stated before, can contain multiple visual concepts that reference the same concepts. All aforementioned concepts, with the exclusion of repositories, can have attributes as stated in section 3.

What is not shown in fig. 7 is the relation to notation where each concept is related to the metamodel of the model, being the notation. This makes every concept an instance of the type defined in the notation meta-level. Models can be impacted by changes in the notation that are already referenced. Corrections on the notation should not impact these models as long as the semantics are equal. New versions of notations, e.g. ArchiMate 2.0 to 3.0 and 3.1, could impact models when the user wants to upgrade. Model elements that refer to a deleted notation element pose a challenge. We will propose a solution to this challenge in another paper.

Though easily interpreted differently, the connections between elements are not restricted to the same model. We can connect model elements by connections from another model, and even from a model in another repository. This enables the connection of reference models to the models that users create. Though not implemented yet in the user interface, we want to reference models in three ways. Referencing models can be done from inheritance, where we allow changes in our models, effectively following the reference model in all their changes. The next option is the advice, where we can subscribe to changes but will not be forced to change our model if it is not wanted. The last option is the copy, where we make a copy of the reference model and go our separate ways. Each option can be facilitated by connections between the reference model and the derived model, where the connections are stored in an intermediate model to keep track of changes and history. We want to support all three, and we are currently researching the technical consequences of these options.

Another challenge we are currently facing is model reference connection ownership. When two models, model A and model B, contain references to one another. The main question is, who is the owner of mentioned references? In the case of an export of model A, we need to decide what will happen with the references to model B. Because connections cannot exist without elements, the only solution is to either export model B elements or none of the connections. This functionality of model reference connections will come in handy when the modeller wants to

connect the models.

Models have the option to be transformed into a model of another notation. There will be two types of transformations. The first type of transformation is what we call live transformation. Live transformation continuously transforms your model. This means that any changes made in the original model will show in the transformed model. This will be done through an intermediate model with concept-specific transformation information on its connections. The first time the transformation happens, the user will be prompted to provide extra information regarding transforming elements or adding new elements that do not exist in the original notation. This information is saved in the aforementioned intermediate model. The model can be transformed without further user interaction when concepts are edited. When something new is added to the original model, the user will again be prompted to provide the additional information necessary to transform the model. The second type of transformation is what we call a single transformation. In this type of transformation, no intermediate model will be maintained, and changes made in the original model will not carry over to the transformed model. Extra information needed for this transformation will be requested at the moment of transformation, and if the user wants to transform the original once more, it will be requested again. Remark that we still have to decide on the transformation standard that we are going to support.

6. Instances

The lowest level of our metamodel is the instance level. Where the notation is the type for the model, the model is the type for the instance. In the modelling process, one often starts at the instance level where the business can explain this information to the analyst, e.g. during an interview. In modelling DEMO, one uses the PIF analysis to create a model from a narrative. Neither this PIF analysis nor any part of the methodology describes how to register the instances of the modelling information. The methodology does describe the way of working in the colouring and annotation of the narrative but does not specify how to register the instance for the fourth sapience 'verification by instantiation' [17, fig 12.2] and leaves the reader with a few non-integrated examples.

The practical necessity can be found in the example shown in fig. 8. This example [18] shows the model and some instances of a DEMO model and time instances. For the explanation of the content of this figure we refer to the source presentation. We only have added the figure to show an example of how instance representation is attempted. The example is not consistent between the model and the instances. To help provide instances of these types of complex models, a new solution needs to be created.

In FBM, one explicitly starts with the sentences and models from that standpoint. Our Simplified platform will support that way of modelling soon.

We believe that a modelling tool should not only reflect the thoughts of the modeller but should also capture the origin of that model. Therefore, we have built, in the previous iteration of the platform, the narrative behaviour that supports a single instance in the form of loose, small sentences. A user could upload a narrative which was then split into sentences. An element was then created for the narrative and for each of the sentences. These sentences were linked to model elements, which made the text and the elements connect, thus forming traceability of analysis

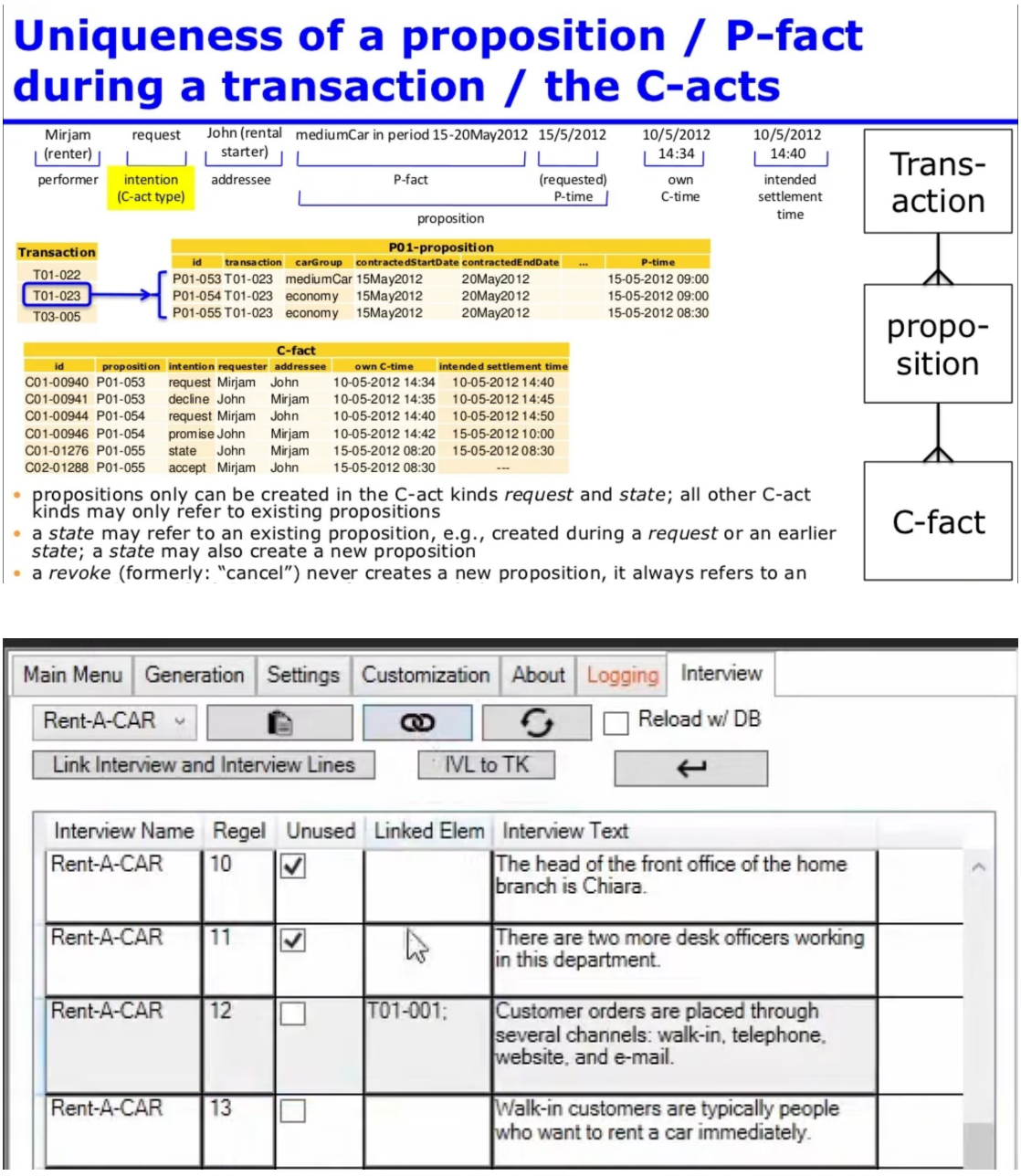

Figure 9: Narrative linking Elements

results. Every narrative sentence can be seen as an instance (a part of) of the model element. The traceability was further enhanced with a button in the UI that would find all linked elements, providing a one-click way of finding the results of a sentence.

In the next iteration of the platform, we want to fine-grain this functionality to the linking of single words to all types of elements throughout the model. Combined with the filtering capacity of the new platform, this would be representable in a specialised overview, such as a diagram with

only linked elements and their words.

Attempts have been made to pre-select words from the narrative on keywords to predict the linking of the words to certain elements. In the context of DEMO, one can think of the word 'request' that will likely be linked to the start of a transaction. For BPMN, a lot of verbs will be connected to an activity. Because natural language is rarely ordered the same way the model is organised, words across the narrative should be grouped before they get linked.

We already have made instance representation with the gamification of instances. We can play the model with play scripts that are instances of the models. These scripts will be generated by the platform and will also be executable in the gamification module of the platform.

One way we can visualise this is by having elements act as diagrams. This allows the user to open the diagram and create instances of this element visually by dragging instance elements onto the diagram. Subsequently, data for this instance can be added as properties of the instance.

Although there is no defined order to create instances, certain instances will depend on other instances and, if made before those other instances, they might have to be revisited to add the said reference. None of the modelling languages provides any information on the way of providing examples and connecting them to the model elements.

We expect that the workforce can better understand models when accompanied by examples. In consultancy assignments, models always need to be explained by examples, and that explanation is what we want to add to the instance level of the platform. This assumption is yet to be investigated. People often can't read the models but can follow the examples. If we can expertly verify that examples are correct and are in line with the model and that people can understand them, then the model is accurate.

7. Conclusion

We can conclude that our progress in notation and model level has advanced far enough to be able to describe the test case notations and create models. This proved that the chosen architecture is sufficient to model a subset of all notations. The instance level needs more research and implementation to be evaluated as usable. We have seen that the simplified platform architecture is divided into six functional layers, split over two servers. Next, we have spoken about the metamodel, and its essential three concepts, element, connection and attribute. In addition, the visual concepts mirror the concepts. The architecture of notations was laid out and showed a meta^4 structure.

Several possible future research subjects have already been discussed and are summarised below.

Firstly, the model level has challenges in ownership of elements, connections, and the semantic value of those elements and connections and aspects of as elements instantiation, composition, subsets, or inheritance of properties, and matching of definitions of other platforms. Secondly, at the instance level, both storage and connectivity pose a challenge that must be built and tested in practice. Next, the dependency handling of individual data entries in the cache layer must be added. Furthermore, the cube and its multi-dimensional textual representation of a part of a model needs to be defined. In addition, what requires more attention is the possibility of extending the structure of notations with the process of creating the notation in such a way that the platform and

users support a complete methodology and are guided in building models. Model elements that refer to a deleted notation element pose a significant challenge. Same holds for the support of all three model reference methods with the technical consequences. Also, the solution to the way of providing examples and connecting them to the model elements will also have to be investigated. Lastly, based on review comments, we have to research more optional relevant work as published in [19, 20, 21, 22, 23, 24, 25, 26] Also other existing tooling like Eclipse, EMF need to be covered in a comparison paper. The research method will be a substantial part of new papers. The idea of notations that can be used and re-defined by other notations seems to be like the concept of profiles in UML and should be described.

References

[1] M. Op 't Land, H. A. Proper, M. Waage, J. Cloo, C. Steghuis, Enterprise Architecture – Creating Value by Informed Governance, The Enterprise Engineering Series, Springer, Heidelberg, Germany, 2008. doi:10.1007/978-3-540-85232-2.

[2] H. A. Proper, M. Op 't Land, Lines in the Water – The Line of Reasoning in an Enterprise Engineering Case Study from the Public Sector, in: A. F. Harmsen, H. A. Proper, F. Schalkwijk, J. Barjis, S. J. Overbeek (Eds.), Practice-Driven Research on Enterprise Transformation – Second Working Conference, PRET 2010, Delft, The Netherlands, November 11, 2010. Proceedings, volume 69 of *Lecture Notes in Business Information Processing*, Springer, Heidelberg, Germany, Delft, the Netherlands, 2010, pp. 193–216. doi:10.1007/978-3-642-16770-6.

[3] L. J. Hommes, The evaluation of business process modeling techniques, TU Delft, Delft University of Technology, 2004.

[4] L. J. Hommes, Modelworld; http://modelworld.nl, 2015.

[5] M. A. Mulder, R. Mulder, F. Bodnar, M. van Kessel, J. Gomez Vicente, et al., The simplified platform, an overview, Modellierung 2022 Satellite Events (2022).

[6] M. Mulder, Enabling the automatic verification and exchange DEMO models, Ph.D. thesis, Radboud University Netherlands, 2022.

[7] J. L. G. Dietz, Enterprise Ontology – Theory and Methodology, Springer, Heidelberg, Germany, 2006.

[8] J. Dietz, H. Mulder, Enterprise Ontology: A Human-Centric Approach to Understanding the Essence of Organisation, Springer Nature, 2020.

[9] J. Dietz, J. Hoogervorst, A. Albani, D. Aveiro, E. Babkin, J. Barjis, A. Caetano, P. Huysmans, J. Iijima, S. J. V. Kervel, The discipline of enterprise engineering, International Journal of Organisational Design and Engineering 3 (2013) 86–114.

[10] M. Mulder, H. Proper, On Enterprise-Grade Tool Support for DEMO, 2021.

[11] S. d. Kinderen, K. Gaaloul, H. A. Proper, On transforming DEMO models to ArchiMate, in: I. Bider, T. A. Halpin, J. Krogstie, S. Nurcan, H. A. Proper, R. Schmidt, P. Soffer, S. Wrycza (Eds.), Enterprise, Business-Process and Information Systems Modeling – 13th International Conference, BPMDS 2012, 17th International Conference, EMMSAD 2012, and 5th EuroSymposium, held at CAiSE 2012, Gdańsk, Poland, June 25-26, 2012. Proceedings,

volume 113 of *Lecture Notes in Business Information Processing*, Springer, Heidelberg, Germany, 2012, pp. 270–284. doi:10.1007/978-3-642-31072-0_19.

[12] R. Ettema, J. L. Dietz, Archimate and demo–mates to date?, Advances in Enterprise Engineering III 34 (2009) 172–186.

[13] L. O. Meertens, M.-E. Iacob, L. J. Nieuwenhuis, M. V. Sinderen, H. Jonkers, D. Quartel, Mapping the business model canvas to archimate, in: Proceedings of the 27th annual ACM symposium on applied computing, ACM, 2012, pp. 1694–1701.

[14] S. de Kinderen, K. Gaaloul, H. A. Proper, Integrating value modelling into archimate, ????, pp. 125–139. EP-2012-Mehdi-IESS.

[15] M. Snoeck, Enterprise information systems engineering, The MERODE Approach (2014).

[16] M. Mulder, R. Mulder, F. Bodnar, Towards a demo description in simplified notation script, in: Enterprise Engineering Working Conference, to be published.

[17] 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, Heidelberg, Germany, 2020. doi:10.1007/978-3-030-38854-6.

[18] M. O. 't Land, Implementation of ontological business transactions using demo and the normalized systems approach, 2012. URL: https://www.slideshare.net/MartinOptLand/implementation-of-ontological-business-transactions-using-demo-and-the-normalized-systems-approach-op-t-land-ist-16-jul2012-final.

[19] H.-G. Fill, D. Schremser, D. Karagiannis, A generic approach for the semantic annotation of conceptual models using a service-oriented architecture, International Journal of Knowledge Management (IJKM) 9 (2013) 76–88.

[20] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendovszky, Á. Lédeczi, Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure., MPM@ MoDELS 1237 (2014) 41–60.

[21] J. Liu, S. Zschaler, B. Baudry, S. Ghosh, D. Di Ruscio, E. Jackson, M. Wimmer, Joint proceedings of models'13 invited talks, demonstration session, poster session, and acm student research competition, CEUR-WS. org, 2013.

[22] S. Kelly, J.-P. Tolvanen, Collaborative creation and versioning of modeling languages with metaedit+, in: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2018, pp. 37–41.

[23] H.-G. Fill, D. Karagiannis, On the conceptualisation of modelling methods using the adoxx meta modelling platform, Enterprise Modelling and Information Systems Architectures (EMISAJ) 8 (2013) 4–25.

[24] R.-H. Pfeiffer, A. Wąsowski, Texmo: A multi-language development environment, in: European Conference on Modelling Foundations and Applications, Springer, 2012, pp. 178–193.

[25] H. Kern, A. Hummel, S. Kühne, Towards a comparative analysis of meta-metamodels, in: Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, 2011, pp. 7–12.

[26] R. F. Paige, N. Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G. K. Olsen, S. Zschaler, Rigorous identification and encoding of trace-links in model-driven engineering, Software & Systems Modeling 10 (2011) 469–487.