Validating the DEMO Specification Language

M. A. T. Mulder(✉)

Leusden, Netherlands
mark@mulderrr.nl

Abstract. The Design and Engineering Method for Organisations (DEMO) is the principal methodology in Enterprise Engineering (EE). The Design and Engineering Method for Organisations Specification Language (DEMOSL) states the rules, legends, and metamodel of DEMO. Therefore, any DEMO model must comply with this specification. Moreover, to enable automation of the DEMO model validation, we need a metamodel that can accurately represent DEMO models. With DEMOSL as the appointed specification language for DEMO, with automation as target, we need to validate the fitness of DEMOSL for modelling DEMO.

Our findings provide insight into the amount of changes and the complexity and direction of change to complete the metamodel and make it usable for automation. We found that some incomplete, inconsistent or inadequate specifications in DEMOSL hinder its use as a prescriptive metamodel. We describe these limitations in DEMOSL as a whole and in the separate Construction Model (CM), Process Model (PM), Action Model (AM) and Fact Model (FM).

Finally, we conclude that the metamodel needs improvement to be able to model all allowed DEMO models.

1 Introduction

The Design and Engineering Method for Organisations (DEMO) [1] is the principal methodology in Enterprise Engineering [2]. This so-called essential model of an organisation is the integrated whole of four aspect models: the Construction Model (CM), the Action Model (AM), the Process Model (PM) and the Fact Model (FM). Each model is expressed in one or more diagrams and one or more cross-model tables.

1.1 Aspect Models

The CM is the first and the most comprehensive model to produce when modelling an organisation in DEMO, applying the Organisational Essence Revealing (OER) method. A CM is a model of the construction of an organisation (or better: of a Scope of Interest), by which is understood the identified transaction kinds and the actor roles that are either executor or initiator of these transaction

D. Aveiro et al. (Eds.): EEWC 2018, LNBIP 334, pp. 131–143, 2019.
https://doi.org/10.1007/978-3-030-06097-8_8

kinds. The resulting 'network' of transaction kinds and actor roles is always a set of tree structures, which arise from the inherent property that every transaction kind has exactly one elementary actor role as its executor (and vice versa), and that every actor role may be initiator of none, one or more transaction kinds.

A CM is expressed in an Organisation Construction Diagram (OCD), a Transaction Product Table (TPT) and a Bank Contents Table (BCT). The OCD is a graphical representation of the identified transaction kinds and actor roles, and the links between them. Apart from initiator and executor links, actor roles may also be connected to transaction kinds through information links. They express that the actor role has (reading) access to the history of all transactions of the transaction kind with which it is connected. Therefore, the transaction kind shape may also be interpreted as a transaction bank.

The AM of a Scope of Interest comprises the guidelines that guide actors in doing their work, i.e. performing their coordination acts and their production acts. An AM is expressed in Action Rules Specifications (ARSs) and Work Instruction Specifications (WISs). Action rules, which are actually (imperative) business rules, guide actors in responding to the coordination events that they have to deal with. They are currently expressed in a semi-structured English-like language. Work instructions guide actors in performing production acts, i.e. in bringing about the products of transactions.

The PM of a Scope of Interest bridges its CM and the coordination part of its AM. To this end, it specifies how the transaction kinds in a tree are related to each other. More precisely, it specifies which transaction steps in an enclosed transaction kind are connected to which steps in the enclosing transaction kind, and by which kind of link (response link or wait link). A PM is expressed in a Process Structure Diagram (PSD), and (optionally) in one or more Transaction Pattern Diagrams (TPDs).

The FM of a Scope of Interest bridges its CM and the production part of its AM. To this end, it specifies the various entity types, property types, attribute types and entity types, as well as their mutual relationships. The current version of DEMO is called DEMO-3. It is published in [3] and in [4].

1.2 Problem Statement

We started this research on the observation of limited automated support of the DEMO modelling. We considered building new automation with the necessary modelling and validation support. To stick as closely as possible to the design of DEMO we chose to adopt the Design and Engineering Method for Organisations Specification Language (DEMOSL) documentation as the backbone of the automation and hoped to have all rules, legends, and the metamodel of DEMO. This first attempt at a metamodel that can accurately represent DEMO models turned out to have omissions. This triggered the investigation of the DEMOSL information. The underlying research question is to find the metamodel that supports the full representation of DEMO models in such a way that the automation and exchange of valid DEMO models is supported.

1.3 Observations

With the partial DEMOSL-metamodels in mind, we tried to project existing DEMO models onto the metamodel using newly developed automation and in doing so we discovered some practical imperfections. In addition, during the project of building a tool to support DEMO modelling the analysis of the metamodel showed some inconsistencies. Not all partial metamodels allowed for modelling correct DEMO models and not all parts of the metamodel were connected. This may have consequences for the usability of DEMO models as a whole. Theoretical benefits might not be usable when the DEMO model is reduced to its aspect models. DEMO claims to be a method that can reveal the essence of the organisation by combining the four partial analysis. A search of the literature did not yield any research or findings on this subject.

1.4 Research Design

The research was conducted using Design Science Research (DSR) [5]. It aims at the design of a complete and automated metamodel of DEMO, facilitating model validation and integration of other technologies. The first step in the research was to validate DEMOSL to check its fitness as metamodel for this purpose. The second step will be the expansion of DEMOSL to support the specification of automated storage and exchange of DEMO by expanding the specification to all model aspects that need to be described. This not only includes the data model and business rules, but also the representation metamodel needed to exchange the representations of a DEMO model. The third step will be the validation of this metamodel with existing DEMO models. Finally, we will investigate the usability of this automation for building new DEMO models together with other technologies.

In the remainder of this paper we will discuss the DEMO metamodel by first defining the notion of Meta Model. We will subsequently report on our findings on the metamodel per aspect model. We end with conclusions and future research.

2 Notion of Metamodel

To be able to validate the metamodel of DEMO, we need a definition to describe the requirements of the metamodel. We will use the following definition of a metamodel [6]: “meta-models define sets of valid models, facilitating their transformation, serialization, and exchange.”.

Analysing this definition, we have to define which models are valid. The metamodel of these valid models needs to be sufficiently complete to describe all sets of models that are allowed. Moreover, the metamodel needs to be restrictive enough to reject models that are not valid.

According to this definition, metamodels should facilitate exchange of the models. Following this part of the definition means that all aspects of the model

that are drawn, noted, related or specified need to be described in such a way that the same representation can be reproduced with the specification.

The serialisation of a model allows for the exchange of model information with possibly a different syntax than the model is represented itself. This is needed to store and retrieve models and business rules to and from repositories.

Transformation of the models is something that is not included in the base of DEMO but is part of the current research project. Therefore, we must be able to specify the DEMO models in the metamodel well enough to enable the partial transformation of the model (e.g. from ontological to implementation level).

3 DEMOSL Inconsistencies

The consistency of the current metamodel of DEMO, named DEMOSL, was analysed with new tools. Earlier, Gouveia [7] verified the FM and suggested changes in the value type but those did not affect the metamodel. Van Kervel [8] has created his own model to describe specifically the transaction pattern of DEMO models. Other literature research did not yield any results on changes to the metamodel. Furthermore, no documented or published, complete implementation of DEMOSL has been found. In the next paragraphs we will analyse the DEMOSL models. We used the published DEMOSL versions 3.6 [9] and 3.7 [4] to analyse the metamodel. DEMOSL 3.7 has been published and already included some results of this research project.

3.1 Metamodel

We will make three remarks on the model exchange requirement. First, no part of the DEMOSL metamodel is currently capable of registering all needed information to exchange names, formulations or visual information from diagrams, tables or specifications. DEMOSL has never been intended to describe these aspects of the meta model. Secondly, although the name of an object can be seen as the identity of that object, the metamodel must include the name to be able to exchange this identity. Finally, concerning the diagrams and tables used in DEMO, no metamodel is given in DEMOSL. This definition is needed to be able to exchange the same visual information.

3.2 Construction Model

The metamodel of the CM 3.6 (Fig. 1) [9, sl. 27] shows five entity types and represents the meta level to combine actor roles and transaction kinds. In addition, the metamodel of the CM 3.7 (Fig. 2) [4, sl. 31] shows six entity types. The change in the metamodel is a partial result of this research. We will elaborate on the improvements and remaining issues.

The Transaction Kind (TK) entity type models transaction kinds. Aggregate Transaction Kind (ATK) is the aggregation of transaction kinds. With the 'TK is part of ATK' relation, the transactions within a selected set can be replaced

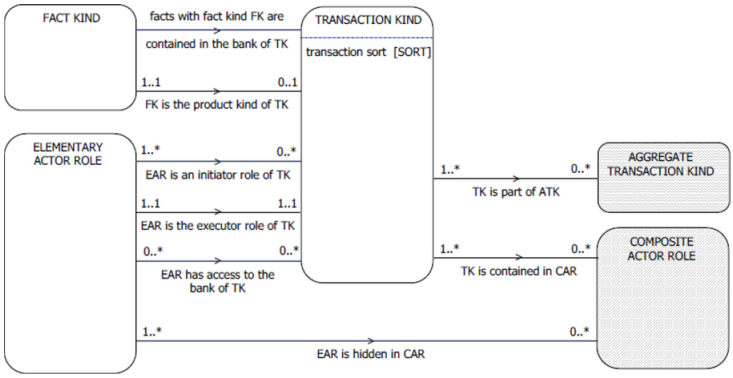

Fig. 1. Meta model 3.6 of the CM [9, sl. 27]

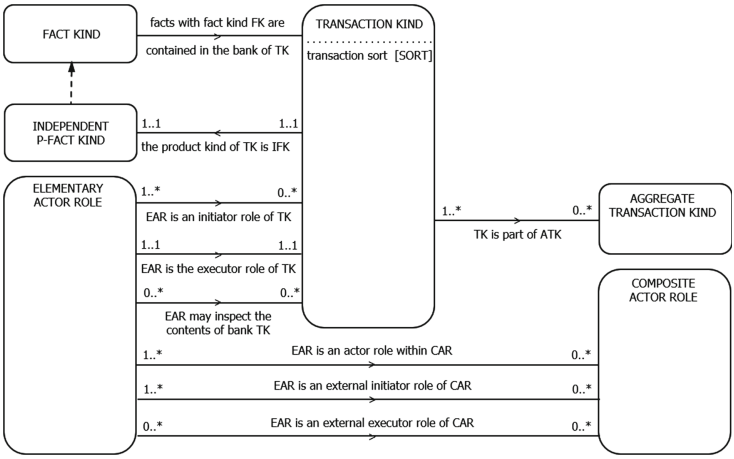

Fig. 2. Meta model 3.7 of the CM [4, sl. 31]

by a single component, an ATK. That is how the transaction kinds stay in the model and are represented in aggregated form. ATK that are out of the Scope of Interest (SoI) have no TK to refer to. The metamodel states that this reference is mandatory (1..*) whereas the example in [3, p. 89] shows several unreferenced instances of ATKs.

The Fact Kind (FK) entity type in DEMOSL 3.6 represents both the coordination fact and production fact of an TK. This notion is not consistent with the FM, therefore, it has been changed in DEMOSL 3.7. The FK and Independent P-Fact Kind (IFK) contain all facts that are created during all transactions. Interstriction between transaction kinds and actor roles is modelled using the 'EAR has access to the bank of TK' that has been rephrased to 'may inspect the contents of bank' relation. Solely inspection of the ATKs cannot be represented in the metamodel.

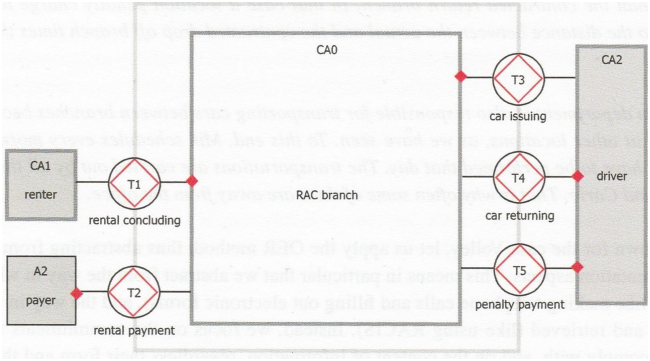

Fig. 3. RAC model from TEoO [3, p. 72]

As can be seen in Fig. 3 the diagram displays a grey boundary. This boundary has been described in 3.6 [9, sl. 7] and 3.7 [4, sl. 7] as the SoI. The DEMO method states that TK can be on the boundary of the SoI to represent a TK communication between the inside and outside of the SoI. The 3.7 metamodel cannot relate any component as being inside or outside any SoI nor can it model the name of the SoI.

The entity types Elementary Actor Role (EAR) and Composite Actor Role (CAR) store the actor roles of the model.

The initiator and executor roles are separately modelled as relations for the elementary and the composite actor roles. This set of relations has evolved from the 3.6 metamodel where 'EAR is hidden in CAR' did not suffice for modelling various EAR roles within a CAR. A collection of actor roles is available through the 'EAR is an actor role within CAR' relation. This relation requires that for every CAR at least one EAR is present. When modelling an actor role outside the SoI, this is not valid.

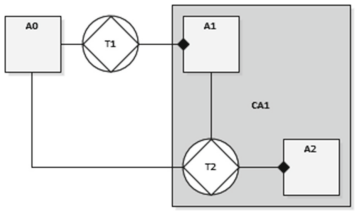

Fig. 4. Transaction kind and CAR

Although the CAR includes TKs according to note 4 [4, sl. 31], the relation 'TK is contained in CAR' of version 3.6 has been omitted and, therefore, the transaction cannot be explicitly related with a CAR. Moreover, note 4 states that the TKs between the EARs are included within CAR. The situation of CAR2 in Fig. 5 and CA1 in Fig. 4 shows that this situation cannot be modelled. In the latter A0 would initiate CA1. Without this metamodel relation it is not clear whether the transaction kind T2 in Fig. 4 is part of CAR.

In the DEMO method, the first step is creating a CM is to create a CAR and relate it to the border transactions. These CARs with their border transactions cannot be captured by the metamodel. The next step in the DEMO method is to reveal the inner actor roles within the CAR. When these actor roles are too

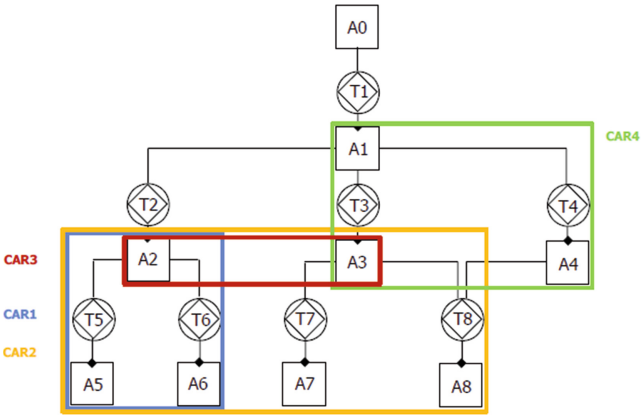

Fig. 5. CAR example [4, sl. 10]

complex to be revealed at once, new CARs can be used. This hierarchical CAR can neither be modelled in metamodel 3.6 nor 3.7. When a EAR is found to be part of multiple CARs, the CAR might relate. This relation is not present in the metamodel and this needs to be resolved. In the metamodel of 3.7, an example (Fig. 5) of multiple CARs within the same CM has been given. CAR1 as well as CAR3 is contained within CAR2[1] but whether this relation is relevant cannot be modelled.

We summarise the issues found in the CM: inconsistent relation to the FM (but fixed in 3.7); mandatory TK for ATKs; mandatory EAR for CARs; missing inspection relation to separate ATKs; missing scope of interest; missing CAR to TK relation; missing CAR hierarchy; missing relation TK in CAR; missing interstriction between ATK and CAR or SoI.

3.3 Process Model

The PM metamodel 3.6 (Fig. 6) shows three entity types. The PM metamodel 3.7 (Fig. 7) shows four entity types. The entity type Transaction Process Step Kind (TPSK) is partially renamed to Transaction Kind Step Kind (TKSK) but we will refer to this entity type with TPSK. The transaction kind entity type represents the same entity type as show in the CM (Fig. 2), making the connection between the CM and the PM.

The Process Step Kind (PSK) entity types in the metamodel 3.6 extends the TK with the process steps using a Cartesian product. This allows for a modelling of the relation between a TPSK of one TK and a PSK in another TK using the 'is initiated from' relation. Moreover, the wait conditions between two process steps can be modelled using 'is a wait condition for' relation.

[1] The initiator and executor roles in the example notes are incorrect.

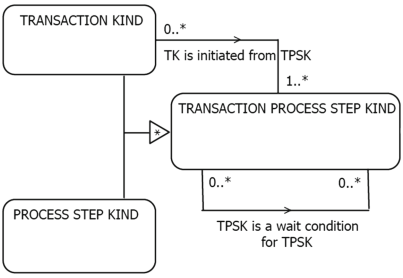

Fig. 6. Process meta model 3.6 [9, sl. 28]

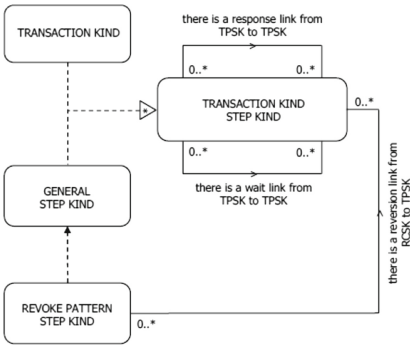

Fig. 7. Process meta model 3.7 [4, sl. 32]

The relation 'TK is initiated from TPSK' in metamodel 3.6 did not allow a TPSK to invoke a Revoke Step Kind (RSK) in another TK. Therefore, in metamodel 3.7 the relation has been changed to a self-reference 'there is a response link from TPSK to TPSK'. This relation does not sufficiently restrict the model and needs note 2 [4, sl. 32] that the transaction pattern will limit the possibilities.

The transaction pattern itself has been extended with a reversion link to support the step kinds from Revoke Pattern Step Kind (RPSK) patterns to the General Step Kinds (GSK). The reversion link is part of the transaction pattern. When the internal transaction pattern is added to the metamodel, all steps should be included. However, the metamodel does not allow for modelling the complete transaction pattern.

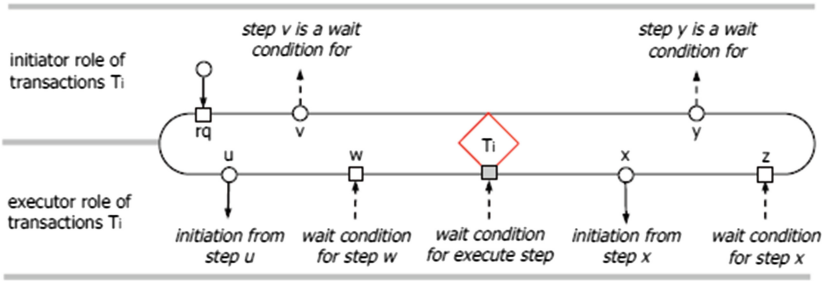

Fig. 8. Process model legend [4, sl. 18]

In the PSD visualisation (Fig. 8), swim lanes are introduced. These are not present in the metamodel.

In summary the issues found in the PM metamodel are: insufficient links between TPSK instances (but fixed in 3.7); partially modelling of transaction pattern; swim lanes.

3.4 Action Model

The AM metamodel (Fig. 9 [4, sl. 33]) has a single entity type representing action rule information of a specific PSK with a reference to the related TPSK. In the formal specifications the AM is specified in Extended Backus - Naur Form (EBNF) [10]. The specification mentions three main parts. The first part specifies the preconditions to execute the action. The second part specifies the conditions to be evaluated. The last part specifies the executed actions in case of a valid or invalid condition. The syntax of the relations to the FM are not fully specified. In the AM it is not clear whether the relation is about reading, writing or creating a FK.

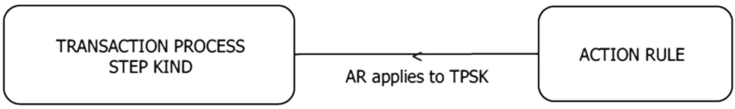

Fig. 9. Action meta model 3.7 [4, sl. 33]

The information in the AM is not sufficiently detailed to validate a model. The verbalisation used in the ARS can be specified in relations (Fig. 10).

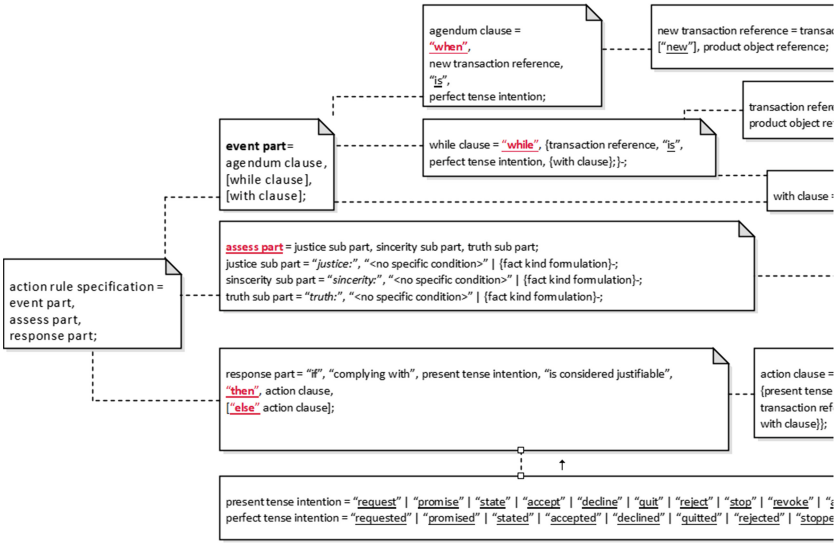

Fig. 10. (Partial) Action rule specification

We analysed the ARS with ANother Tool for Language Recognition (ANTLR) to validate the completeness of the specifications. Though ANTLR

has a slightly different syntax on EBNF, we translated all rules to be validated. The result is that the following specifications were missing: property kind name; object variable; dimension; perfect tense sentence; product object reference. These specifications need to be defined, or their reference needs to be altered, to be able to use the ARS. Furthermore, we found that some definitions of variables were not distinct enough to be parsed by the ANTLR specifications. This could mean that the specification is ambiguous in variable definition. The attribute variable, abstract variable and product variable are connections to the fact model. These connections should be present in the metamodel in Fig. 11.

A summary of the issues found in the AM metamodel are: specifications need additions and elaborations and variable naming needs addition.

3.5 Fact Model

The model and the metamodel are expressed in the same notation. Therefore, we **bold** for the metamodel of the FM and *italic* for the FM itself. The FM metamodel 3.7 (Fig. 11) contains eight **entity type**s. The difference between the 3.6 and the 3.7 version is the addition of the 'P-' naming prefix. We omit this prefix in this paper for short writing where no ambiguity can be found. This metamodel allows for simple *entity type*s and for specialised, generalised or aggregated *entity type*s. *Entity types* can contain two types of *property type*s: *value type* (i.e. attribute) and *property type*s (i.e. relation).

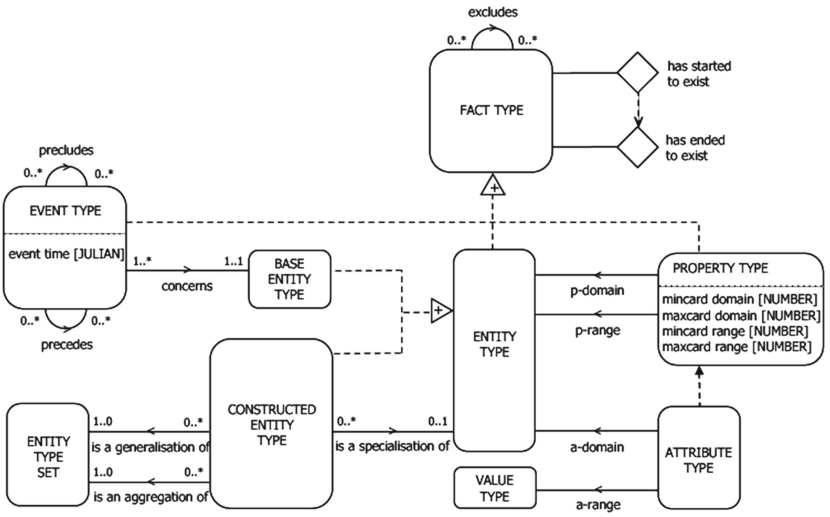

Fig. 11. Fact model 3.7 metamodel [4, sl. 34]

An *entity type* can be modelled using **entity type**. The name of an *entity type* [4, sl. 22] has not been added to the FM metamodel.

The *property type*, or relation, between *entity type*s is expressed as a **property type** that has a domain **entity type** and a range **entity type**.

$$pdomain(x) \iff x \in \textbf{entity type}$$

$$prange(x) \iff x \in \textbf{entity type}$$

$$\exists x : relation(x) \implies x \in \textbf{property type}$$

$$\forall x, y : relation(x, y) \implies pdomain(x) \wedge prange(y)$$

The name of a *property type* [4, sl. 22] has not been added to the FM metamodel.

The **attribute type** is a specialisation of **property type** and acts the same way as the **property type**. Note 2 [4, sl. 34] states that the relations adomain and arange are also specialisations. An attribute can be formulated in this rule

$$adomain(x) \iff x \in \textbf{entity type}$$

$$arange(x) \iff x \in \textbf{value type}$$

$$\exists x : attribute(x) \implies x \in \textbf{attribute type}$$

$$\forall x, y : attribute(x, y) \implies adomain(x) \wedge arange(y)$$

The name, dimension and unit of an *attribute type* [4, sl. 22] has not been added to the FM metamodel.

Every **property type** has a minimum and maximum cardinality for domain and range.

The **event type** entity type matches the IFK entity type of the CM. Therefore, it adds two relations to the creation of a *fact kind* in the CM; precedes and precludes. The precedence law [4, sl. 26] states that two **fact kind**s have an order in time. The preclusion law states that the two **fact kind**s cannot occur both. This precedence as well as the preclusion law affect the CM and the PM but are not mentioned in either the metamodel or the diagrams of the CM or the PM. The *property type* event time on the **event type** entity type is probably meant as the moment in time the event occurred. This is also expressed in the 'has started to exist' event type.

The concerns relation links the **event type** to the **base entity type** in such a way that every transaction kind that creates an independent fact kind also has to have a concerning **entity type**. This is expressed on instance level in this rule

$$\forall x : event(x) \implies baseEntity(x)$$

$$\exists x : baseEntity(x) \implies event(x)$$

This notion of concerns to a **base entity type** means that an **event type** can never concern a **constructed entity type**. Therefore, transaction kinds cannot be a concern to specialisations of entity types whereas in the example RAC [3, p. 75] the 'RENT-PAID RENTAL' is a specialisation of 'RENTAL' while 'RENT-PAID RENTAL' is the concern of 'P2 the rent of Rental is paid'.

The **constructed entity type** deals with the specialisation of **entity type**s and the generalisation and aggregation of **entity type set**s. The distinction between type and type set is used to keep together the entity types used for the set operation. Note 3 [4, sl. 34] does non visually connect the **entity type set** to the **entity type**. We only found a typo '1..0' in the relations of the **constructed entity type**.

The **fact type** is a generalisation of **entity type** which can start or end to exist. To model the existence of a fact, we need the time it began its existence and the time its existence ended. Occurring production events will be stored as an **event type**, similar to the **property type**, related to an **entity type**.

The exclusion rule on **fact type** [4, sl. 25] can be used to make two disjoint collections of **entity type**s, **property type**s or even **event type**s.

$$\textbf{entity type1} \cap \textbf{entity type2} = \varnothing$$

$$\begin{aligned} property1(x) &\iff x \in \textbf{entity type1} \\ property2(x) &\iff x \in \textbf{entity type2} \\ \forall x, y : property1(x, y) &\implies \neg property2(x, y) \\ \forall x, y : property2(x, y) &\implies \neg property1(x, y) \end{aligned}$$

In summary the issues found in the FM metamodel are: no name on entity type; no name on property type; no name, dimension and unit on attribute type; CM and PM lack precedence and preclusion laws; specialisations cannot be concerned with event type; cardinality on entity type set is incorrect; no time in start and end events.

4 Conclusions and Future Research

When comparing the aspect models with our requirements for automated support we can conclude that the metamodels are incomplete, inconsistent and, moreover, not implementable in their current state. This restricts the possibilities for automating DEMO model validation. Despite the required improvements DEMOSL is a solid base for a metamodel. The CM, PM, FM and AM all have some faults in their metamodels, therefore, we are unable to implement a correct representation of a DEMO model without changes. To validate all models a complete metamodel with all business rules has to be build. This supersedes the current entity type metamodel of the models but requires a complete and restricted mathematical model of all allowed DEMO models.

For automation there is a need to improve the metamodel DEMOSL to be able to model all allowed DEMO models. This metamodel needs all entity types to address the properties and relations between all concepts. In addition, the metamodel needs to be completed with all business rules to create a full DEMO specification.

Post presentation comments on the paper have been addressing the relevance of making a distinction between the concrete and abstract syntax of the metamodel. The discussion on the different concepts of SoI and CAR that essentially are addressing the same problem will be part of the next paper. Relevant is the usage of the metamodel. In future research the intended use of the metamodel needs to be addressed first. Finally, the usage of the DEMOBAKER AM syntax will be studied for usage in the specification of the AM.

References

1. Dietz, J.L.G.: Enterprise Ontology: Theory and Methodology. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-33149-2
2. Dietz, J.L.G., Hoogervorst, J.A.P.: The discipline of enterprise engineering. Int. J. Org. Des. Eng. **3**, 86–114 (2013)
3. APC Perinforma: The essence of organisation. Sapio Enterprise Engineering (2013)
4. Dietz, J.L.G., Mulder, M.A.T.: Demo specification language 3.7 (2017)
5. van Aken, J., Andriessen, D.: Handboek ontwerpgericht wetenschappelijk onderzoek. (Handbook for Design Science Research). Boom Lemma (2011)
6. Aßmann, U., Zschaler, S., Wagner, G.: Ontologies, meta-models, and the model-driven paradigm. In: Calero, C., Ruiz, F., Piattini, M. (eds.) Ontologies for Software Engineering and Software Technology, pp. 249–273. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-34518-3_9
7. Gouveia, D., Aveiro, D.: Things, references, connectors, types, variables, relations and attributes – a contribution to the FI and MU theories. In: Aveiro, D., Pergl, R., Gouveia, D. (eds.) EEWC 2016. LNBIP, vol. 252, pp. 181–195. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39567-8_12
8. Van Kervel, S.J.H.: Ontology driven enterprise information systems engineering. PhD thesis, Delft University of Technology, Delft (2012). ID: urn:NBN:nl:ui:24-uuid:8c42378a-8769-4a48-a7fb-f5457ede0759; ths:Dietz, J.L.G. - org:TU Delft - dgg:TU
9. Dietz, J.L.G.: Demo specification language 3.6 (2017)
10. Feynman, R.: EBNF: a notation to describe syntax (2016)