Towards the Ontological Foundations for the Software Executable DEMO Action and Fact Models

Marek Skotnica[1(✉)], Steven J.H. van Kervel[2], and Robert Pergl[1]

[1] Czech Technical University, Prague, Czech Republic
skotnicam@gmail.com, robert.pergl@fit.cvut.cz
[2] Formetis, Boxtel, The Netherlands
steven.van.kervel@formetis.nl

Abstract. The discipline of enterprise engineering and the DEMO methodology enable a model-driven approach to enterprise software systems development.

Apart from the graphical notation, the DEMO models may be fully specified in the DEMOSL language, which may become a basis for an workflow software system implementation. However, the current specification of DEMOSL has been designed mostly for the reasoning between human stakeholders.

In this paper a formal calculation construct called a DEMO Machine is proposed and basic ontological foundations of this machine are elaborated based on the alignment with the theories of enterprise engineering, various ontological and formal quality criteria and the application of the Generic Systems Development Process for Model Driven Engineering (GSDP-MDE methodology).

Keywords: Enterprise engineering · DEMO · DEMO machine · Enterprise operation system · Ontological foundations

1 Introduction

The domain of this paper encompasses enterprises, ontological foundations and enterprise information systems. Enterprises, as defined in [1], are social systems composed of human actors communicating about their productions to serve some external entity, typically called the "customer". An enterprise is an engineering artefact, designed and implemented for a specific purpose. An enterprise information system [2] is an information system (IS) that provides (i) some valuable descriptive perspective on the operation of an enterprise, for example: financial, personnel, inventory, production monitoring systems; or (ii) executes a prescriptive role, a control system that steers the operation of an enterprise, driven by the execution of a model of that enterprise. An example of such a prescriptive control system is a workflow system.

D. Aveiro et al. (Eds.): EEWC 2016, LNBIP 252, pp. 151–165, 2016.
DOI: 10.1007/978-3-319-39567-8_10

For the engineering of enterprises and the supporting information systems (ISs), appropriate scientific foundations are provided by the DoEE [1] and other theories, as described in Sect. 2. A promising approach is to derive IS directly from conceptual models of enterprises, eliminating manual programming, i.e. a manual translation of software specifications into propositions of a computer programme expressed in a programming language. This approach is proposed and generally elaborated by the Model-Driven Engineering [3].

The relevance of the MDE approach to derive ISs is very high, given that the majority of IT projects in the professional world fails or are "challenged" [4,5]. There is no evidence available that MDE is a panacea to this situation, however the MDE approach based on the DEMO methodology is scientifically interesting due to its strong theoretical foundations (Sect. 2) and a good empirical degree of appropriateness [6–8].

DEMO models are engineering specifications with the C4-ness qualities (models are Concise, Coherent, Consistent and Comprehensive [9]). The approach to implementation of ISs directly derived from DEMO models [10] is a novel topic. However, published professional results [7] document feasibility of this approach that addresses several serious problems:

(i) The elimination of programming due to the fact that the DEMO model is the source code for a native DEMO model executing software engine.
(ii) A substantially better business-IT alignment, enabled by automatic model verification, early model validation, followed by incremental model improvements, eliminating most of the human errors induced by manual programming.
(iii) Reduction of complexity due to highly expressive specifications and abstraction layers.

The paper is organized as follows: In Sect. 2, the underlying scientific foundations are briefly discussed. In Sect. 3, the research question is more precisely defined. In Sect. 4, the axioms of the theory are proposed, investigated and represented in a formal notation. In Sect. 6, the related work is discussed. In Sect. 7, the current results are summarised and further research is mentioned.

2 Scientific Theories and Methodologies Applied

We understand the notion of ontology as a "formal, explicit specification of a conceptualization shared between stakeholders" [11]. The most important criteria regarding the quality of any ontology [12] are (i) ontological truthfulness, an ontology providing a truthful representation of the real world; (ii) ontological completeness, completeness of expression for any phenomenon that may exist in our domain of the real world; and (iii) ontological appropriateness, good support for shared reasoning between stakeholders. The mentioned C4-ness quality criteria must be also met.

Other important concepts regarding ontologies, conceptual languages and models are formulated by Guizzardi in [13].

Enterprise Ontology [9] is a theory for enterprises, composed of human actors that communicate and cooperate about some production fact, to meet the requirements of some external entity, typically called a "customer".

The DEMO methodology [9] is an engineering methodology, based on the theory of enterprise ontology, to devise conceptual models of enterprises.

The notion of the model quality is represented by the already mentioned C4-ness criteria, as well as three cardinality laws [10,13,14].

MDE essentially means using ontologies for designing artefacts, which is the domain of The Design Science Paradigm described by [15], who provides a framework to devise engineering artefacts.

The Normalised Systems Theory formulates rules that software systems[1] must adhere to, to be evolvable and maintainable over time [16].

The Generic Systems Development Process for Model Driven Engineering (GSDP-MDE) [10,14] provides a methodology to devise a conceptual language and a model-executing software system based on a domain ontology. The GSDP-MDE is a special case of the Generic System Development Methodology, which is based on the General Systems Theory [17]. GSDP-MDE provides a model-instance driven approach where for each phenomenon instance in the real world, there is one unique instance of a model that is a precise *descriptive* representation of the specific phenomenon instance in the real world. At the same time, the model instance provides a *prescriptive* representation of the allowed state transitions of this real world phenomenon instance. The model-instance driven approach is the foundation for the descriptive and prescriptive ISs.

3 Formulation of the Research Question

3.1 The DEMO Machine Concept

The currently latest version of the DEMO 3 specifications of models and representations [18] is based on DEMOSL, an acronym of DEMO Specification Language. DEMOSL is specified using EBNF (extended Backus-Naur Form) [19]. This language is derived from reasoning on rules and facts from a modeller's perspective, primarily used for shared reasoning between stakeholders. The main research question is: **How should a DEMO Machine be designed that can be used to interpret DEMOSL?** Such a machine should take as its input a specification based on DEMOSL and enhanced by necessary execution semantics. As this is a very broad topic, we will limit ourselves here to discussing just a fragment of this machine, as specified further. Moreover, the DEMO Machine is meant as a formal computation model (similar to the e.g. Turing Machine), i.e. we do not elaborate on software implementation.

3.2 Appropriateness of the DEMOSL for DEMO Machine Implementation

The DEMO Machine needs to take into account the following challenges that are induced by the execution level and thus not addressed in DEMOSL:

[1] But not limited to.

1. *Integration.* DEMOSL concepts are either new (to be carried out) or existing concepts already present in the enterprise. DEMOSL does not deal with this separation.
2. *Facts duplication.* The facts representation must be physically present in one place to assure the consistency of enterprise systems [20]. This point is related to *Integration*, but it is also valid on its own.
3. *Lack of expressiveness.* At the execution level, there are many domains, where DEMOSL is not expressive enough to describe it, like scientific computations.
4. *Modularity.* DEMOSL does not specify, how the solution is modularised, which is at the same time a crucial execution concern.
5. *Lack of version transparency.* DEMOSL does not deal with evolvability of the models with the respect to the running instances.
6. *Execution semantics of DEMOSL.* Currently, the execution semantics of DEMOSL is not fully specified. Let us demonstrate this on a simple example:

```
when T01(M) is requested with member(new M) = P
  if age(P) < minimal_age then decline T01(M)
  else promise T01(M)
```

The following semantics is not defined:

- How `with` should be executed?
- What does `age(P)` mean, where and how should it be calculated?
- What does `minimal_age` mean, where and how should it be stored?

This list of topics is probably not complete, however it names the key challenges that definitely need addressing.

3.3 Formulation of Ontology for DEMO Machine

Based on these observations, we narrow our research question to: How to formulate a domain ontology for Facts, Agenda and Rules (FAR)? We started building the DEMO Machine ontology from this topic, because facts, agenda and rules are the "heart" of a DEMOSL execution. This ontology should address the points listed above and it should exhibit the necessary qualities:

(i) To be based on and compliant with the FI, TAO and PSI theories of EE.
(ii) Truthfulness and good appropriateness qualities and compliance with the three cardinality laws [13].
(iii) Maintaining the strict C4-ness criteria.

As for the notions of "Way of thinking" and "Way of working" distinguished in the DEMO method and theories, the FAR ontology is rather a way of working, as we come from the existing way of thinking (the PSI theory) and formulate how to enhance it for the execution.

3.4 Verification and Validation Questions of the Research Question

The FAR domain ontology is proposed below that enables the construction of executable Fact and Rule expressions that operate on Agenda. As for the model verification, we must make sure that it is free of anomalies that may enable a construction of rules and expressions that cannot exist in the world of phenomena.

4 Axiomatic Specifications of the Fact, Agenda and Rule Ontology

4.1 Addressing the DEMOSL-DEMO Machine Deficiencies

Let us elaborate, how the FAR Ontology (as well as the whole DEMO Machine) may address the challenges stated above.

1. *Integration* and *Facts duplication*. Based on the *Separation of Concerns Principle* from the Normalised Systems Theory [16], the DEMO Machine should not supply the functionality of the already-existing enterprise systems, such as a database. Also, the DEMO Machine should not specify scales, dimensions, sorts, units such as time, money and others.
2. *Lack of expressiveness*. For areas, where there are already established solutions (like mathematical libraries), these should not be represented in a DEMO Machine, to maintain the separation of concerns and the C4-ness criteria.
3. *Modularity* and *Version transparency* are complex topics that cannot be easily commented. They are a subject for future work that should be based on the studies of Normalised Systems Theory mentioned above.
4. *Execution semantics of DEMOSL*. The execution semantics should be specified by the DEMO Machine. The FAR Ontology focuses on the subset of execution, namely the facts, agenda and rules concepts.

Let us now dive into the specific part of the DEMO Machine, the FAR Ontology, which will be specified as a set of axiomatic definitions.

4.2 Fact Axioms

The DEMO theory builds on the Φ theory. The letter Φ stands for "FI", an acronym for Fact and Information about a "world", being a specific part of the universe we are interested in, and of which we require factual information or knowledge [9]. Our world of interest is "the world of enterprises". A world of interest is assumed to be composed of Acta, Facta and Stata. **Stata** are things or phenomena that existed before the beginning of our observation. A **Fact** is a proposition about something that exists in the real world and provides us with *factual knowledge* about the world. Facts can be about either *concrete* or *abstract* things or phenomena. They are the results of **Acta**, being actions or acts, undertaken by an entity. Facts come to being by carrying out acts. Once they originate, they cannot disappear; they can be only ignored.

During the design time, we deal with facts as *propositions* about the real world. They exist just as a symbolic structure and we cannot decide its truthfulness. Then, once the DEMO Machine executes (i.e. the fact "happens"), we may *valuate* it as `true`, `false` or `undefined`. Undefined means that the subjects of the proposition does not exist, yet, or we do not know the valuation, as a result of e.g. a technical failure. The valuation may (and typically does) change during the execution. Any calculations based on facts should take this into account. Stata also represent factual knowledge about our world of interest that exist since the beginning of time. Obviously any facts about Stata are always either true or false.

Let us present the definitions here using the standard mathematical constructs.

Definition 1. ***Fact*** *A fact is an ordered tuple:*

$$Fact := (Identifier, Type, Proposition) \tag{1}$$

Identifier – A unique identifier of the fact.
Type $\in \{Internal, External, Composed\}$
Proposition – A specification[2] *of the statement about the real world.*

Definition 2. ***Value of a fact*** *is a valuation function:*

$$FactValue : (TransactionInstance, Fact) \rightarrow \{True, False, Undefined\} \tag{2}$$

Definition 3. ***Transaction Instance Linking (TIL)*** *is a ternary relation that relates certain transaction instances to each other. This relation is defined outside of the DEMO Machine, which requests this relation for the evaluation of the rules.*

$$\begin{gathered} TransactionInstanceLinking(TIL) := \\ TransactionInstance \times TransactionInstance \times LinkingIdentifier \end{gathered} \tag{3}$$

TransactionInstance – A transaction instance unique identifier.
LinkingIdentifier – A name of the relation that holds between the transaction instances.

Example 1. Two transaction instances are sharing the same membership:
("T01_1","T02_2","Membership")

Definition 4. ***Internal Fact*** *is a factual statement about a DEMO model instance.*

$$InternalFact := (Fact, InternalFactExpression) \tag{4}$$

[2] FAR does not specify the language, it may be a natural language or any other language.

Definition 5.

$$InternalFactExpression := (singleTransactionComparison)|(multiTransactionComparison) \tag{5}$$

singleTransactionComparison = (transaction).state (operator) ((transaction).state | (state))
multiTransactionComparison = (transactionSelector).(selectorFunction) (t => (singleTransactionComparison))
transaction = this | this.parent
state = perfect tense intention" as defined in DEMOSL
operator = == | !=
variable = (transaction).(attribute)
selectorFunction = all | any
transactionSelector = transactionType < (*linkingIdentifier*) > |
this.children < *transactionType* >
transactionType = existing transaction type defined in the model
linkingIdentifier = identifier of the relation between transactions

This grammar is using the Extended Backus-Naur Form (EBNF). Round brackets denote non-terminals. Note that the presented grammar is very basic and it is not able to capture all facts about the DEMO model instance or its history. Complete grammar is a subject for further research.

Example 2. Let us show an example by formalising the fact F02 "Are invoices paid?", which is the situation when all instances of T03 that are linked to the current transaction are in the same state as the current transaction.

F02 = (("F02","Are invoices paid?"), T03< "*Invoice*" >.all(t => t.state == this.state))

Definition 6. ***External Fact*** *is a Fact about the world outside the DEMO Machine*

$$ExternalFact := (Fact, CalculationEngine) \tag{6}$$

CalculationEngine – Identifier of the external system function evaluating the fact.

Data in external data banks are represented as external facts, for instance. External facts represent knowledge of phenomena in the environment that may change over time and have no (known) calculation specification. We operate just with a further unspecified reference to external system function that is able to valuate the fact, thus carrying out the separation of concerns principle.

Example 3. A fact that evaluates that the person attached to the transaction instance is older than 18 years

F01 = (("F01","Is person older than 18 years?"), CalculationEngine)

CalculationEngine may be implemented in any computer technology such as a web service (SOAP or REST), or locally as a system library. In the following code, we implement it as a class in a standard programming language. The calculation of a F01 would be realized as its method:

```
public class CalculationEngine {
[DEMOEngineExternalFact(FactId="F01")]
public FactValue IsPersonEligible (TransactionInstance t) {
var person = DAL.GetPersonByTransactionInstanceId(t.Id);
if(person == null) return FactValue.Undefined;
else return person.Age > 18 ? FactValue.True : FactValue.False;
}}
```

Definition 7. ***Composed Fact*** *is a fact composed from internal and external facts.*

$$ComposedFact := (Fact, ComposedFactExpression) \tag{7}$$

Definition 8. ***Composed Fact Expression***

1. *InternalFactIdentifier and ExternalFactIdentifier are composed fact expressions.*
2. *If x and y are composed fact expressions, then following expressions are also composed fact expressions:*
   (a) (x and y)
   (b) (x or y)
   (c) not (x)

For valuation of composed facts, the Kleene and Priest three-valued logics is used.

Example 4. A person is older than 18 years and he is accepted as a applicant in a membership approval process of the Volley tennis club:

```
F01 = (("F01","Is person older than 18 years?"),
  VolleyCalculationEngine)
F02 = (("F02","Is person accepted in the approval process?"),
  VoleyCalculationEngine)
F03 = (("F03","Is person eligible for membership?"),
  ("F01 and F02"))
```

The resulting truth table is then:

F01	F02	F03 Result
True	True	True
True	False	False
True	Undefined	Undefined
False	True	False
False	False	False
False	Undefined	False
Undefined	True	Undefined
Undefined	False	False
Undefined	Undefined	Undefined

4.3 Agenda Axioms

An agenda is set of possible coordination acts (agendum) that is presented to the actor. These are well-defined concepts in the PSI theory. An actor involved in a transaction is offered, according to the transaction axiom, to choose one of the valid options to perform coordination acts, which happens in asynchronous time. Example: After a Request from the initiator, the executor may issue either a Promise or a Decline, but other coordination acts such as a Reject are now forbidden, to comply with the Transaction Axiom.

An agenda for an actor must be (re)calculated completely at run time by the DEMO Machine of the model instance, after each state change of the model instance. It will be shown that the allowed options for coordination acts are restricted by causal and conditional dependencies and rules. It means that application of rules is present to guarantee the compliance with the PSI theory. Any extension, enlargement, of the transaction transition space or the state space is impossible since this would violate the PSI theory axioms.

Definition 9. ***Coordination Act (cAct)*** *is a proposed or intended action for an actor.*

$$cAct := (Transaction, TransactionInstance, \\ ActorInstance, Intention, SettlementType) \tag{8}$$

Transaction = Transaction kind as defined in DEMOSL.[3]
TransactionInstance = Associated transaction instance. May be empty.
ActorInstance = Associated actor instance.
Intention $\in$ {*Create(T, n), Promise, Decline, Request, Quit, Accept, Reject, State, Stop, RevokeRequest, AllowRevokeRequest, RefuseRevokeRequest, RevokePromise, AllowRevokePromise, RefuseRevokePromise, RevokeState, AllowRevoke State, RefuseRevokeState, RevokeAccept, AllowRevokeAccept, RefuseRevokeAccept*}
SettlementType $\in$ { *Allow, Enforce, Restrict* }

There are two additions to the definition given by the DEMO theory. One is the possibility to create a new transaction (generated by the composition axiom) which will be used by the rules. $Create(T, n)$ means "Create n transactions of type T", where n is a positive whole number. The second is the settlement type which says how the cAct should be dealt with. `Allow` means that an actor is allowed to perform the intention. `Enforce` cAct says that the given intention should be actually performed, unless there is a `Restrict` cAct with the same intention for the same transaction instance. Practically, the `Restrict` cAct also informs the actor, why such an intention cannot be performed. In the DEMO theory, an actor is allowed to perform an act even when it is restricted. However, in the enterprise practice, legal and other compliance is a crucial aspect of execution. Thus, we enable this feature in the DEMO Machine.

[3] Transaction is also defined by the TransactionInstance if present.

Please note that in Definition 9 we do not take into account any additional information from inside or outside of the organization. This is due to the separation of concerns principle addressing the Facts duplication (Sect. 4.1). All external information (facts) are handled outside of the DEMO Machine.

Definition 10. ***Agenda*** *is a function that calculates a set of actor's possible actions based on the current state of the model taking into account the composition axiom and the respective rules.*

$$Agenda : (ModelInstance, ActorInstance) \rightarrow \{cAct\} \tag{9}$$

Definition 11. ***Perform cAct***

$$PerformCAct : (ModelInstance, ActorInstance, CActToEnforce) \rightarrow Agenda \tag{10}$$

To perform a cAct means that the actor makes a selection of an allowed cAct from its agenda and it enforces it.

4.4 Rules and Dependencies Axioms

Rules and dependencies are specifications of either a prescriptive execution of a coordination act, or a conditional prohibition of a coordination act for an actor, depending on the evaluation of a fact.

A rule and a dependency restrict the available freedom of an actor to issue coordination acts at the execution time. The evaluation, if the rule or dependency applies, takes place at runtime, depending on the state of that model instance. The transaction instance state space and the state transition space of a model instance is further restricted (made smaller). It is impossible to add new options for coordination acts since that would violate the axiomatic specifications derived from the PSI theory.

Definition 12. ***Causal Rule and Dependency*** *are defined as the application of a rule that results in a transaction state change.*

$$CausalRule = (Transaction, TransactionState, Fact, cActTrue, cActFalse) \tag{11}$$

Definition 13. ***Evaluation of Causal Rule and Dependency***

```
if TransactionInstance.State == TransactionState
  and FactValue(TransactionInstance, Fact) == True
then anAgenda.Add(cAct(cActTrue, Enforce))
else if False then anAgenda.Add(cAct(cActFalse, Enforce))
```

Definition 14. ***Conditional Rule and Dependency*** *are defined as the application of a rule that results in a restriction of an agendum, in such a way that one of the allowed coordination acts is prohibited while the rule applies.*

$$ConditionalRule = (Transaction, Fact, cActToRestrict) \tag{12}$$

Since facts may change over time during execution, a condition that inhibits a specific cAct can be met, and the specific cAct is permitted. If one of two cActs is prohibited in the agenda, then the opposite cAct can be performed in asynchronous time by the actor. As long as the fact in the conditional rule holds, it is not possible for the actor to perform the cAct.

Definition 15. ***Evaluation of Conditional Rule and Dependency***

```
if anAgenda(TransactionInstance).Contains(cActToRestrict(Allow))
  and FactValue(TransactionInstance, Fact) != True
then anAgenda.Add(cActToRestrict(Restrict))
```

Prohibition or Prescription of an Agenda. From the above follows that rules and dependencies operate on an agenda by prohibition or prescription. They reduce the model instance state space and the model instance transition space, which causes a desired limitation of complexity. It is impossible to increase the state and transition spaces by "adding" new options for coordination acts which would be violation of the PSI theory. Rules and dependencies are calculated immediately during the calculation of the agenda.

5 Discussion and Evaluation of the FAR Ontology

The relation between the FAR Ontology and the DEMO models is as follows. The DEMO models provide a formal specification of the rules and facts, created and accepted by stakeholders, that represent the enterprise interaction with its environment. The DEMO Machine specifies the construction of an artefact (a software system) that must fulfil the requirements of the created DEMO models. The FAR Ontology is a crucial part of the DEMO Machine.

The following reasoning is provided to assure:

(i) A compliance with the PSI theory, the causal and conditional dependencies, and the application of explicitly specified causal and conditional AM rules.
(ii) A reduction of complexity while maintaining guaranteed ontological conciseness and comprehensiveness.

Assume a model composed of actors and transactions. The application of the Transaction Axiom reduces the number of states of each transaction and the number of states in the model state space, which results in a reduction of complexity.

The application of the Composition Axiom demands that before any production fact can be performed, all child production facts must have been produced, i.e. Stated and Accepted. This further reduces the number of states in the model state space. The ontological conciseness and comprehensiveness of the PSI theory has been shown in [2].

The application of the causal dependencies reduce the state transition space of the model instance, since a specific option of an agenda must be chosen, while the other agenda options are forbidden.

Conditional dependencies disable specific agenda options, until a specific condition has been met. In this way, the state transition space is reduced further and the state space is also reduced, without a loss of ontological conciseness and comprehensiveness.

The DEMO Action Model conditional and causal rules modify the agenda similarly to causal and conditional dependencies and they reduce the state space and the state transition space further, without any loss of ontological conciseness and comprehensiveness.

For a DEMO Machine based solely but precisely on the PSI theory, it has been argued and shown that there is minimized expression, or zero entropy in expression quality [14]. Any enterprise that may possibly exist in the real world can then be represented by one and only one model. In addition, anything that is not an enterprise cannot be represented. Based on this reasoning, it is argued that such DEMO Machine based on a proper implementation of the FAR Ontology will keep these qualities.

5.1 Falsifiable Proposition of the FAR Ontology

As any domain ontology is a hypothesis that provides falsifiable propositions about the world of phenomena [21]. The following hypothetical assumptions have been made:

1. "The PSI theory is a domain ontology, a falsifiable hypothesis about the world of coordination between actors". There is much empirical evidence for a good degree of confidence in the ontological truthfulness and appropriateness of the PSI theory. The C4-ness qualities have been proven. The construction of the DEMO engine using the GSDP-MDE has been proven.
2. "Any business rule that may exist in the real world can be expressed in restriction(s) of actor agenda, by conditional or causal rules". This proposition is directly derived from the PSI theory, hence with a good confidence.
3. "Any fact that can be defined in the real world may be used in a rule to express any business rule in an enterprise". As the facts are either Internal or External, which is a complete list of fact types in the world, we may assume that any given rule may be expressed using a Composed Fact.
4. The hypothesis that any imaginable fact represented using the DEMOSL representation can be expressed in an appropriate way using the FAR Ontology, needs further theoretical research and empirical validation.

6 Related Work

6.1 Model-Driven Development

Model-Driven Development (MDD) is a very popular approach in the recent years realising the ideas of Model-Driven Engineering for implementing software systems. It is a software development approach based on modelling and transformations [22]. The product to be developed is described using various types

of models specifying the requirements, functions, structure and deployment of the product. These models are used to construct the product using transformations between models and code generation. MDD was originally based on Model-Driven Architecture (MDA) developed by the Object Management Group (OMG) [23] defining these types of models: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), Implementation Specific Model (ISM).

The most usual part of the MDD approach is the process of forward engineering to represent the transformations of more abstract models into more specific ones. The most common use-case of such a process is the development of conceptual data models and their transformation into source codes or database scripts.

Our approach shares the idea of driving the development by models. DEMOSL models represent CIM, DEMO Machine and the FAR Ontology brings in execution semantics, which may be related to PIM. However, our approach is to directly interpret the PIM model. In this respect, our approach is similar to the following effort.

6.2 XModel

The solution of devising a workflow software system based on model presented by Johanndeiter et al. in [24] is based on the OrgML modelling language, a part of the MEMO framework, and the XMF metaprogramming platform. The idea is also based on applying the MDE approach, while avoiding the error-prone manual coding stage. The idea is based on applying multiple levels of metamodelling and utilising XMF's unique features to support multiple dynamic levels of abstraction. The approach seems very interesting, however it seems to lack a proper evaluation in enterprise. Our approach also differs in a careful selection of ontologically well-founded methodologies that exhibit necessary qualities and benefits, as discussed in Sects. 2 and 5.

6.3 The DEMO Engine and the Enterprise Operating System

DEMO Engine of the ForMetis Consultants company is a software system for designing DEMO models with the ability to simulate DEMO models for validation and to provide model execution in full production [7]. Construction of DEMO models is done using the graphical representation of the DEMO ATD in a graphical environment. In the current implementation, the DEMO Process Model is primarily calculated from the ATD. Response links and waiting links (causal and conditional dependencies) can be then specified using the graphical representation of the PSD. There is a limited and not well-engineered support for even simple Action Model rules, which is the aim of our FAR Ontology.

The Enterprise Operating System [10] is software system composed of a set of DEMO models and a DEMO model executing software engine, the DEMO Engine. The EOS captures and controls all phenomena that occur in operation of the organizational business transactions. This is very similar to an operating system of a computer that reads from and writes to binary registers of a CPU

and peripheral controllers and supports many tasks. Using a computer without an operating system is extremely difficult and error-prone. This seems to apply also to controlling and monitoring enterprises without an appropriate enterprise operating system. Many of the engineering challenges of this effort are directly related to the DEMO Machine and open questions described in Sect. 3.

7 Conclusions and Further Research

We proposed the concept of a DEMO Machine as a theoretical construct for DEMO models execution. We then proposed the FAR Ontology as a key part of the DEMO Machine. The proposed further research topics are present in the respective parts of the paper.

The FAR Ontology using the GSDP-MDE approach for model-driven information systems provides an approach for enterprise information systems implementation with considerable benefits (Sect. 3). Some of the concepts have been already implemented in DEMO Engine described in Sect. 6.3.

As for future work, the remaining parts of the DEMO Machine need to be formulated, so that every DEMOSL model may be executed. This comprises an exact formulation how the Transaction and Composition Axioms are applied for the model execution. The work should address the concerns named in Sect. 3. As for the FAR Ontology itself, algorithms with proper qualities implementing the proposed functions need to be elaborated and a broader empirical research on the appropriateness in the professional world is suitable. A single empirical business case with inappropriate expressiveness would invalidate our hypothesis and provide valuable clues for improvement of the FAR Ontology.

References

1. Dietz, J.L.G., Hoogervorst, J.A.P.: The discipline of enterprise engineering. Int. J. Organ. Des. Eng. **3**(1) (2013)
2. Guerreiro, S., Kervel, S., Babkin, E.: Towards devising an architectural framework for enterprise operating systems. In: Proceedings of ICsoft 2013 8th International Conference on Software Paradigm Trends. SciTePress (2013)
3. Bzivin, J., Gerb, O.: Towards a precise definition of the OMG/MDA framework. In: IEEE International Conference on Automated Software Engineering (2001)
4. Sauer, C., Cuthbertson, C.: The State of IT Project Management in the UK. Templeton College, Oxford University, Oxford (2003)
5. Budzier, A., Flyvbjerg, B.: Double Whammy, How ICT projects are fooled by randomness and screwed by political intend. In: CRASHH Conference, University of Oxford (2011). Draft v5
6. Mulder, J.B.F.: Rapid enterprise design. Ph.D. thesis, Delft University of Technology (2006)
7. Hintzen, J., van Kervel, S.J.H., van Meeuwen, T., Vermolen, J.A.J., Zijlstra, B.: A professional case management system in production, modeled and implemented using DEMO. In: Proceedings of 16th IEEE Conference on Business Informatics (CBI) (2014)

8. Op 't Land, M.: Applying architecture and ontology to the splitting and allying of enterprises. Ph.D. thesis, University of Technology Delft (2008)
9. Dietz, J.: Enterprise Ontology Theory and Methodology. Springer, New York (2006). ISBN: 3-540-29169-5
10. van Kervel, S.J.H., Dietz, J.L.G., Hintzen, J., van Meeuwen, T., Zijlstra, B.: Enterprise ontology driven software engineering. In: Proceedings of ICsoft 2012 7th International Conference on Software Paradigm Trends. SciTePress (2012)
11. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993). Knowledge Systems Laboratorium, Computer Science Department, Stanford University
12. Guizzardi, G., Ferreira Pires, L., van Sinderen, M.: An ontology-based approach for evaluating the *domain appropriateness* and *comprehensibility appropriateness* of modeling languages. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 691–705. Springer, Heidelberg (2005)
13. Guizzardi, G.: Ontological foundations for structural conceptual models. Ph.D. thesis, University of Twente (2005)
14. van Kervel, S.J.H.: Ontology driven enterprise information systems engineering: Ph.D. thesis, University of Technology Delft (2012)
15. Hevner, A.: A three cycle view of design science research. Scand. J. Inf. Syst. **19**(2), 87–92 (2007). Information systems and Decision Sciences, University of South Floria, USA
16. Mannaert, H., Verelst, J.: Normalized Systems Re-creating Information Technology Based on Laws for Software Evolvability. Koppa, Belgium (2009)
17. von Bertalanffy, L.: General System Theory: Foundations, Development, Applications. George Braziller, New York (1968)
18. Dietz, J.: DEMOSL-specification: version 3.4, CIAO! enterprise engineering. Network (2016). doi:10.5281/zenodo.47471
19. ISO, Geneva: ISO 14977. Information technology Syntactic metalanguage Extended BNF, Norm (1996)
20. Coronel, C., Morris, S.: Database Systems: Design, Implementation, & Management, 11th edn. Course Technology, Cambridge (2014)
21. Popper, K.R.: Zwei Bedeutungen von Falsifizierbarkeit [Two meanings of falsifiability]. In: Seiffert, H., Radnitzky, G. (eds.) Handlexikon der Wissenschaftstheorie (in German), pp. 82–85. Deutscher Taschenbuch Verlag, Mnchen (1994). ISBN: 3-423-04586-8
22. Mellor, S.J., Clark, A., Futagami, T.: Model-driven development. IEEE Softw. **20**(5), 14–18 (2003)
23. OMG: Model driven architecture (MDA): The MDA guide rev 2.0. online. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01
24. Johanndeiter, T., Goldstein, A., Frank, U.: Towards Business Process Models at Runtime, pp. 13–25. MoDELS@ Run. time 1079 (2013)