

A DEMO Machine - A Formal Foundation for Execution of DEMO Models

Marek Skotnica^{1(✉)}, Steven J.H. van Kervel², and Robert Pergl¹

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 provide enterprise designers with a formal techniques to design companies where competency, responsibility and authority is clearly defined. In such companies, process-based anomalies can be avoided and people tend to cooperate more effectively and contentedly.

These techniques are so far mostly used just for business process modeling consultancy. DEMO-based software systems are needed to adopt and support these techniques in professional companies. This paper proposes a theoretical computation concept called DEMO Machine that provides us with formal foundations for a simulation of DEMO models. We demonstrate these formal foundations on a Volley Club example.

Keywords: DEMO machine · Enterprise engineering · DEMO simulation · DEMO software implementation

1 Introduction

The Enterprise engineering community has been working on formal theories and methodologies for more than 15 years. The results were found to surpass the state of the art of business process management (BPM) approaches in terms of formal correctness, ontological completeness, and anomalies [1]. But, so far an adoption of these principles in practice is very slow. One of the reasons is that the largest benefit from these theories is provided to middle-sized or large companies and these organizations tend to change very slowly. In addition, a new technology adoption is associated with high risks. Large IT systems with many complex features are required, as well, usually provided by large companies such as IBM, Pega, Oracle, or Microsoft. There are no such large DEMO-based IT systems so far. As argued in the FAR Ontology paper [2], it is not easy to understand how the DEMO models are simulated. This work builds on van Kervel's work [3], simplifies it according to the Occam's law and enables for further extensions. It also builds on ForMetis company professional experience in building DEMO-based systems.

The goal of this paper is to propose a theoretical computation foundations that are easy to understand (like BPMN) and yet allow to express all the DEMO

aspect models. This is a prerequisite for building DEMO-based IT systems that could compete with state-of-the art BPM systems (BPMS). For this purpose, we propose a DEMO Machine – an abstract formalism, which can be used for DEMO model simulation and DEMO model code implementation.

The paper is organized as follows: In Sect. 2, the research question is summarized. In Sect. 3, the underlying scientific foundations are briefly discussed. In Sect. 4, formal definitions of DEMO Machine are proposed, investigated, and represented in a formal notation. In Sect. 5, the proposed theories are demonstrated on a Volley Club example. In Sect. 6, the current results are summarized and further research is proposed.

2 Research Question

This paper elaborates on a research question proposed in FAR Ontology paper Sects. 3.1 and 3.2 [2]. The DEMO Machine is meant as a formal computation model (similar to the e.g. the Turing Machine). The DEMO Machine needs to take into account challenges that are induced by the execution level and thus not addressed in DEMOSL [2]. The research question was stated as: **“How should a DEMO Machine be designed to interpret DEMOSL?”**.

3 Theories Used and Related Work

Theories used in this paper were already mentioned in the FAR Ontology paper [2], therefore we just offer a brief summary of them: Guizzardi’s ontology theories [4], the Enterprise Ontology [5], the DEMO methodology [5], van Kervel’s papers [3, 6].

This paper is also influenced by related work in this area, most notably: Figueira and Aveiro [7], Huysmans [8], Krouwel [9], and Op’t Land [10].

4 DEMO Machine

This section elaborates on the research question proposed in Sect. 2. To investigate characteristics of a software system, it is better to do it on its formal model rather than on its software implementation. We do take an inspiration from Turing’s invention called the Turing Machine [11], which was the first universal computer made in 1936, years before any physical computers existed.

DEMOSL provides specification for the DEMO models of an enterprise. However, for the simulation of the models there are no definitions provided yet. Therefore, we define the missing concepts and propose a formal DEMO Machine that is able to simulate the models. This machine is independent on any software implementation, and it is only based on the mathematical concepts.

4.1 DEMO Model Definitions

In this section, essential DEMO model definitions are provided in a form that is suitable for DEMO Machine simulation. The semantics of these concepts is aligned with the DEMO theory [5]. DEMO model is an ontological representation of an enterprise. Demo models are commonly represented by four aspect diagrams – OCD, PSD, OFD, and AM. Diagrams together express a DEMO model. The following formalization deals with the DEMO model itself.

Definition 1. Actor Role. *An actor role is an ordered tuple:*

$$ActorRole := (Identifier, ActorRoleType) \quad (1)$$

Identifier – A unique identifier of an actor role.

ActorRoleType $\in \{Elementary, Composite\}$

An elementary actor role is an atomic amount of authority, responsibility, and competence. It is a producer in exactly one transaction, and a customer of zero, one, or more transactions [5]. A composite actor role is a network of transaction kinds and (elementary) actor roles, of which one does not (want to) know the details [12].

Definition 2. Transaction Kind. *A transaction kind is an ordered tuple:*

$$Transaction := (Identifier, TransactionKindName, Executor, Initiators) \quad (2)$$

The second axiom of the Ψ -theory states that coordination acts are performed as steps in universal patterns [5]. These patterns, also called transactions, always involve two actor roles and are aimed at achieving a particular result [5]. These patterns are formally defined in Sect. 4.3.

Definition 3. Causal Link. *A causal link is an ordered tuple:*

$$CausalLink := (SourceTransactionKind, SourceState, \\ TargetTransactionKind, TargetState, MinCardinality, \\ MaxCardinality, InitiatorActorRole) \quad (3)$$

InitiatorActorRole – An initiator Actor Role to distinguish to which executor this link applies since a transaction can have multiple initiators.

According to the theory, a causal link is defined as: “a link between a coordination act and its resulting coordination fact, indicating the fact is result of the act.” [12]. A Causal link is used in a tree-like structure to define a business process composed of multiple transactions. For example, when there is a causal link from T1/pm to T2/rq it means that you can initiate a new T02 instance from a T01 instance that is in state promised or a later state.

Definition 4. Conditional Link. A conditional link is an ordered tuple:

$$\text{ConditionalLink} := (\text{SourceTransaction}, \text{SourceState}, \text{TargetTransaction}, \text{TargetState}, \text{InitiatorActorRole}) \quad (4)$$

InitiatorActorRole – An initiator Actor Role to distinguish to which executor this link applies since a transaction can have multiple initiators.

Conditional link restricts the source transaction state from being reached until the causal link's cardinalities are satisfied. For example, there is a causal link from T1/pm to T2/rq with cardinality 1..1. There is a conditional link from T02/ac to T01/st. This means that you can perform cAct T01/st only when one child transaction T02 reached ac.

Definition 5. DEMO Model. A DEMO Model is an ordered tuple:

$$\text{DEMOModel} := (\text{Identifier}, \text{TransactionKinds}, \text{ActorKinds}, \text{ConditionalLinks}, \text{CausalLinks}, \text{Facts}, \text{Rules}) \quad (5)$$

A DEMO Model is a conceptual representation of an enterprise or a sub-enterprise at a given time frame. Facts and rules definitions are provided in [2].

4.2 DEMO Enterprise Application Definitions

We considered model definitions so far, but once the simulation of a model takes place, the instances need to be taken into the account because they represent the day to day operation of an enterprise.

Definition 6. Enterprise Position. A DEMO enterprise position is an ordered tuple:

$$\text{EnterprisePosition} := (\text{Identifier}, \text{ActorRoles}) \quad (6)$$

Identifier – Is and identifier of DEMO Enterprise Position

ActorRoles – Is a finite set of actor roles. An ActorRole can belong to several Enterprise Positions.

An enterprise position is defined as a coherent set of actor roles. In practice, it means a principle to group these roles and define responsibilities, competence, and authorities at a generic level; e.g. sales director, production manager etc. These are sometimes also called *functional roles*.

Definition 7. Actor. An actor is an ordered tuple:

$$\text{Actor} := (\text{Identifier}, \text{EnterprisePositions}) \quad (7)$$

Identifier – A unique identifier of an actor instance.

EnterprisePositions – A finite set of enterprise positions

An actor is a person or group of persons (board) that operates in an enterprise in given enterprise positions.

Definition 8. Transaction. *A transaction is an ordered tuple:*

$$\text{Transaction} := (\text{DEMOModel}, \text{TransactionKind}, \text{ParentTransaction}, \text{InitiatorActor}, \text{ExecutorActor}, \text{State}) \quad (8)$$

DEMOModel – A model according which a transaction behaves.

TransactionKind – Is a type of transaction.

ParentTransaction – Is parent transaction. May be empty in case of a root transaction.

InitiatorActor – An actor that initiated the transaction.

ExecutorActor – An actor that is responsible for the execution side of the transaction. May be empty or changed over time as the execution responsibility may be delegated.

State – The current transaction state. States are further explained in Sect. 4.3.

A transaction represents an actual situation, in which the transaction kind is carried out (by people).

Definition 9. DEMO Enterprise Application. *A DEMO enterprise application is an ordered tuple:*

$$\text{DEMOEnterpriseApplication} := (\text{Identifier}, \text{PublishedModels}, \text{EnterprisePositions}, \text{Actors}, \text{Transactions}) \quad (9)$$

PublishedModels – Is a finite set of DEMO models.

Identifier – Is an identifier of DEMO Enterprise Application.

EnterprisePositions – A finite set of enterprise positions that actors can participate in. Enterprise positions can only contain actor roles defined in Published-Models.

Actors – Is a finite set of Actors.

Transactions – Is a finite set of Transactions.

A DEMO enterprise application represents an actual enterprise that consists of DEMO models, actors, and their interactions. A DEMO model is a conceptualization of an enterprise in one given time. A real-world enterprise changes over time, and therefore it needs to act according to multiple DEMO models, resp. their versions. For example, a mortgage company creates a contract in 1990 based on certain conditions, and these conditions do still need to apply in 2017 even though conditions for new mortgages are different. A DEMO Enterprise can also consist of multiple sub-enterprises or departments represented by multiple DEMO models. This concept is very important for a software system implementation, it allows aggregation of agenda – work-items from multiple DEMO Model instances.

4.3 DEMO Axiom Definitions

There are three DEMO axioms that need to be formalized and performed in order to calculate an agenda for a given transaction instance. Agenda, and cAct definitions are provided in [2].

Definition 10. *DEMO Axiom* is a function that takes an agenda and calculates a set of cActs:

$$DEMOAxiom : (Transaction, Agenda) \rightarrow \{cAct\} \quad (10)$$

Transaction Axiom. For the purposes of software simulation, we do formally define the transaction axiom as a state machine in Fig. 1. The circles are states (cFacts), and the boxes are allowed actions (cActs). With this state machine, an implementation of Transaction DEMOAxiom function is straightforward.

We propose also some practical changes that make the transaction axiom suitable for building enterprise information systems. We added a possibility to start a transaction instance without being requested. This supports the real-world situations where people start to negotiate about a transaction. Documents are created, but no request has been made, yet. A distinction whether a transaction starts with a request or initiate or both is done as an extra information on the causal link in the PSD.

The second deviation from the theory is that we do not support revoke of all states at all times. This simplification is mostly because of the composition axiom. For example, when a child transaction is created from the promised state, therefore you are not able to revoke the promise. Revokes combined with the composition axiom are quite a challenging topic in the execution and are a subject to further and mostly empirical research.

Composition Axiom. Composition axiom adds cActs based on the conditional and causal links, so that the transaction instances can form a process. An implementation of this axiom is out of scope of this paper, and it is a subject for further research.

Rule Axiom. Rule axiom adds cActs based on the conditional and causal rules based on the definitions from the FAR Ontology [2].

4.4 DEMO Model Simulation

An Operation of an organisation is the manifestation of its construction in time [13]. Simulation is the imitation of the operation of a real-world process or system over time [14]. DEMO model simulation is the imitation of the operation of a DEMO model for a purpose of validation of the model correctness. DEMO model execution is a DEMO model simulation for the purpose of supporting an operation of an enterprise IT system.

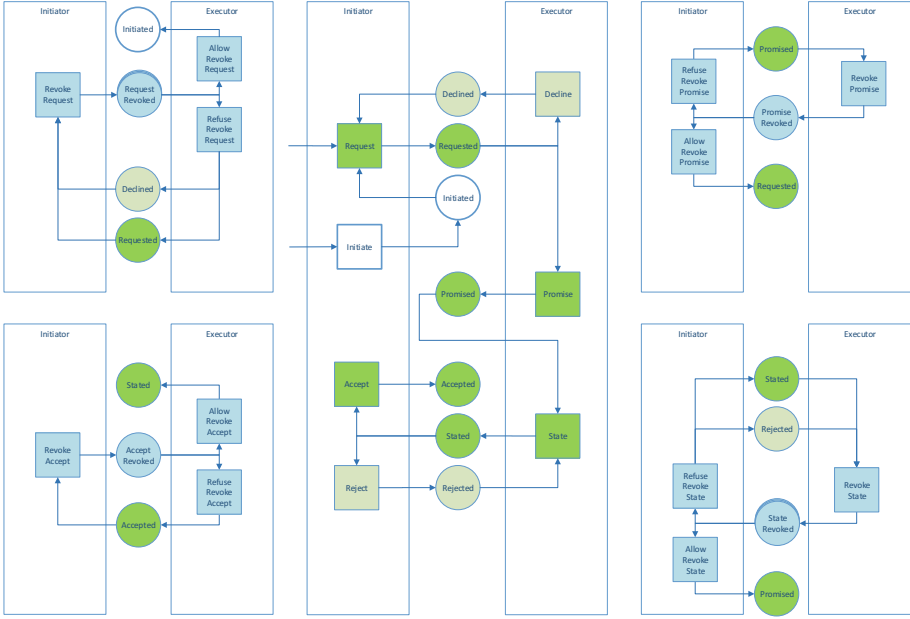


Fig. 1. Transaction axiom state machine

Definition 11. A *DEMO Machine* is an ordered tuple:

$$DEMOMachine := (DEMOEnterpriseApplication, \\ ExternalFactImplementations, TransactionInstanceLinking, \\ InputInstructions, OutputMessages) \quad (11)$$

DEMOEnterpriseApplication – A *DEMO* enterprise application.

TransactionInstanceLinking – Ternary relation that represents connections between transaction instances in the outside world.

ExternalFactImplementations – Outside world implementations of functions that calculate external facts.

InputInstructions – A set of instructions that the machine needs to process.

OutputMessages – Results produced by the machine that represent facts about a behaviour of an enterprise.

The DEMO Machine is receiving instructions on the input and producing messages on the output.

The list of allowed instructions is:

- **GetActorAgenda(Actor)** – Writes an *Agenda* for a specified *Actor* into *OutputMessages*.
- **PerformCAct(cAct)** – Performs a *cAct* and puts a new *Agenda* for the actor instance (defined in *cAct*) into *OutputMessages*. Performing an empty *cAct* causes a recalculation of the model instance.

The Algorithm 1 shows a pseudo-code of how the agenda is calculated for a transaction instance.

Algorithm 1. Agenda calculation

```

1: function CALCULATEAGENDA(transactionInstance, actorPerformCActs)
2:   #Adds actors perform cActs
3:   agenda ← actorPerformCActs
4:   #Adds allowed cActs based on Transaction axiom
5:   agenda.add(TransactionAxiom(agenda))
6:   #Adds allowed and restricted cActs based on Composition axiom
7:   agenda.add(CompositionAxiom(agenda))
8:   #Adds perform and restricted cActs based on Rule axiom
9:   agenda.add(RuleAxiom(agenda))
10:  #Find perform cActs that are allowed and not restricted.
11:  if agenda has cAct c to perform then
12:    #Performs cActs selected to be performed
13:    PerformCAct(transactionInstance, c)
14:    #Transaction states have been changed so recalculation of agenda is
      needed.
15:    return CalculateAgenda(transactionInstance, nil)
16:  else
17:    #No cActs to be performed found, agenda reached a stable state.
18:    return agenda

```

The presented algorithm is just a high-level abstract schema. A detailed description of the DEMO Machine calculation is outside of the scope of this paper.

5 Proof of Concept – Volley Club

In this section, a proof-of-concept DEMO Machine is demonstrated on a Volley club model from the book “The Essence of the Organization” by Jan Dietz [15]. The model is well specified in the book, so we do not elaborate on it much, and we rather point out the differences in our approach and the proposed way of simulation.

To verify the formal definitions, we created a proof-of-concept software implementation of the presented DEMO Machine. In this section we use a general object-oriented pseudo-code inspired by C# to implement the simulation according to the definitions provided above.

5.1 DEMO Model

The organization construction diagram (OCD) in Fig. 2 contains two transactions describing the situation where a customer comes into the club, requests a membership, pays for it, and he becomes a member.

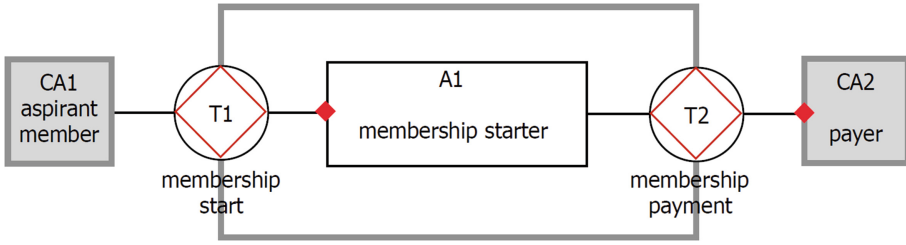


Fig. 2. OCD model volley club [15]

The Process Diagram (PSD) describes how are the two transactions related. The membership payment is requested after a membership start is promised. There is also a conditional link which specifies that the membership execution phase can't be done until the membership payment is accepted. Cardinality is not mentioned here, but we expect only one payment per membership. Later payments are not part of the model.

The Action Model (AM) here consists of four rules, and all of them are for the membership starter (A1). The logic of working with facts defined in the OFD is also included in the rules, but DEMO materials do not elaborate on how they should be dealt with. The precise definition, how to execute this AM rules, is also not provided, but for a communication between human stakeholders, this notation is sufficient.

1. **Action Rule for A1(1)** – When the membership start (T1) is requested, in case the person who is requesting is eligible, then it is automatically promised, otherwise declined. Eligibility means that the person is old enough, starting day of the membership is the first day of some month and maximum number of members was not reached.
2. **Action Rule for A1(2)** – When the membership start (T1) is promised, then automatically request the membership payment.
3. **Action Rule for A1(3)** – When the membership payment (T2) is stated, while the paid amount for the membership has been paid, then accept the membership payment (T2), otherwise reject (T2).
4. **Action Rule for A1(4)** – When the membership start (T1) is promised while the membership payment (T2) is accepted, then execute the membership start (T1) and state the membership start (T2).

5.2 DEMO Machine Model

Here is how the same Volley club model looks like when described by the concepts introduced in this paper.

OCD and PSD remain the same. They are represented as:

```
AspirantMember = ("Aspirant member", Composite);
MembershipStarter = ("Membership starter", Elementary);
Payer = ("Payer", Composite);
T1 = ("T01", "Membership Start", MembershipStarter, {AspirantMember})
T2 = ("T02", "Membership Payment", Payer, {MembershipStarter})
VolleyClubModel = ("Volley Club", {T1, T2}, {AspirantMember,
    MembershipStarter, Payer}, ...)
```

Information about memberships or person is likely to be stored in an external database and there is no use in duplicating them inside the DEMO Machine, as explained in [2].

The action model implementation differs from the DEMO, so let's go through the Volley club business rules and see how they are expressed in the DEMO Machine.

Action Rule for A1(1) is represented by an external fact and a causal rule. The external fact contains all the business conditions that are needed in order to evaluate, whether a person is eligible for a membership. The *LogicalProposition* is there merely to suggest what logic should be used to evaluate such fact. The real logic then lies in the outside world implementation, and it calls the database. A benefit of this approach is that we do not need to change the model when this business rule is modified. A new implementation version is simply plugged in, and the system goes on.

The causal rule *T1RequestedCausalRule* is there to implement the action part (state transition) of the AM rule. It says: "When an instance of *transaction1* is in state Requested and fact *IsMemberElegibleFact* is evaluated as True, then add a cAct with SettlementType=Perform and Intention=Promise to the transaction instance agenda. If the fact is evaluated as False, then add a cAct with SettlementType=Perform and Intention=Decline to the transaction instance agenda." This explanation may seem to be more complicated than the previous action rule, but it covers much more scenarios. Adding of an enforcing cAct is used instead of a direct state transition, because the transition may be forbidden by some conditional rule. The state transition also needs to be allowed by the transaction or the composition axiom. In case of multiple rules enforcing different state transitions, a priority should be assigned to the rules.

```
IsMemberElegibleFact = ExternalFact("Is member eligible for application
?",
    LogicalProposition = "Person.Age >= Minimal_Required_Age",
    VolleyClubCalculationEngineId)
T1RequestedCausalRule = CausalRule(T1, Requested, IsMemberElegibleFact,
    cAct(T1, T1.Current, T1.Current.Executor, Promise, Perform),
    cAct(T1, T1.Current, T1.Current.Executor, Decline, Perform))
```

Action Rule for A1(2) – is represented by a causal rule and an external fact. The external fact will be always True in this case since there are no business conditions. The causal rule is expressed bellow and it says: "If the transaction instance of type T1 is in state Promised and the *TrueExternalFact* is evaluated

as True, then add a cAct that (i) performs creation of a new instance of T2 that will be a child of the current T1 transaction instance and (ii) will be in state Created to the current transaction instance agenda”.

```
T1_Promised_CausalRule = (T1, Promised, TrueExternalFact, cAct(T1, T1.
    Current, T1.Current.Executor, Create(T2, 1), Perform), null)
```

An interesting problem is that the transaction instance T1 can get into state Promised multiple times. Does it mean that it should create a new instance of T2 each time it gets there? And does it depend on some external system? In this model, the creation of unwanted transactions is controlled by the 1..1 cardinality defined in the PSD. However, for generic purposes, we introduced a possibility for external fact implementation to return a number of transactions to be created together with the fact result. This is the way, how we can control how many transactions are created.

Another problem is in determining the executor actor instance for a created transaction instance of T2. It is clear in this particular model that the membership payer will be the same person as an aspirant member. However, it is not formally defined. We do delegate this problem to the outside world implementation. Once it is notified about the created instance of T2, it has all the information it needs to assign the executor. More empirical experience shows, whether this is sufficient, or a more sophisticated solution needs to be designed.

This proof of concept implementation does not contain the composition axiom, and therefore the rules to create child transactions are not implemented, as well.

Action Rule for A1(3) – is represented by a causal rule and an external fact. The external fact is a business rule that determines whether the paid amount was enough. The causal rule then performs accept or reject.

```
IsPaidAmountEnoughFact = ExternalFact("Is paid amount for membership
    enough?",
    LogicalProposition = "this.Membership.AmountToPay <= this.Membership.
        Payment.AmountPaid", VolleyClubCalculationEngineId)
T2StatedCausalRule = CausalRule(T2, Requested, IsPaidAmountEnoughFact,
    cAct(T1, T2.Current, T2.Current.Executor, Accept, Perform),
    cAct(T2, T2.Current, T2.Current.Executor, Reject, Perform))
```

Action Rule for A1(4) – is represented by a conditional rule and a communication fact. We only want the execution phase to be allowed when the child transaction of T1 instance is in state Allowed. We capture such fact using a communication fact that says: “Are all current transaction instance children with type T2 accepted?”. If there is no child transaction with type T2, the fact is evaluated as Undefined.

The conditional rule says: “If there is a cAct with Intention=State and SettlementType=Allow within the current transaction instance agenda and the fact *IsMembershipPaidFact* is not evaluated as true, then a cAct with Intention=State and SettlementType=Restrict is added to the current transaction agenda.” Simply put, the transaction instance state Stated can be only reached when the fact is True.

```

IsMembershipPaidFact = CommunicationFact("Is membership paid?",
    CommunicationFactExpression = "this.children<T02>.all(t => t.state ==
        accepted)", VolleyClubCalculationEngineId)
T2StatedCausalRule = ConditionalRule(T1, IsMembershipPaidFact, State)

```

5.3 Volley Club Outside World Implementation

The outside world consists of the implementation of external facts, transaction relation provider, and state change receiver. It can be implemented in any programming language, as long as it provides values required in the definitions. In our proof of concept implementation, we created a simple implementation of such system that accessed a database and returned relevant values. However, a detailed description of such implementation is not relevant for purposes of this paper.

5.4 Step by Step Execution

In this section, we will provide detailed description of what happens in the execution of Volley club model during the happy-flow scenario.

At first, a Volley club enterprise application is created, and an implementation of the outside world is attached. The Activity Log shows all the changes in the running enterprise application, and we present all steps of the simulation below.

Step 1 – We create enterprise positions and attach them to actor roles. Then we assign actors to enterprise positions. Marek is going to be a Customer, which is an enterprise position with actor roles *Aspirant member* and *Payer*. Elisabeth is going to be an Employee – the Membership starter since she works in the Volley club.

Step 2 – Marek would like to be a member of Volley club, so he initiates a new transaction 1 instance and selects its executor to be Elisabeth. He is prepared to do a request of the membership, but he needs to fill out the starting day. He fills today and performs the request. Because there is nothing to restrict Marek's request, the transaction moves to state Requested. New *Membership* object is created, and it stores the data Marek entered. In state T1 Requested, a causal rule is defined and therefore evaluated. Marek is 27 years old, and that is enough to be a member of Volley club. The causal rule adds enforcing cAct to the agenda, and it moves the transaction to state Promised. In the Promised state, there is a conditional rule that restricts the State from being performed before the membership is paid. The communication fact is evaluated as Undefined, because there is no accepted child T2. No interaction was required from Elisabeth.

```

New transaction T01 was created with name=T01.1.
T01.1:Request:Allow
Initiator of T01.1 performed Request.

```

T01.1:Request:Allow,T01.1:Request:Perform

Fact "Is member eligible for application?" was evaluated as True.

T01.1:Promise:Allow,T01.1:Decline:Allow,T01.1:RevokeRequest:Allow,T01.1:
Promise:Perform

Fact "Is membership paid?" was evaluated as Undefined.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T01.1:State:Restrict

Step 3 – Elisabeth received a request from Marek, and she would like to deliver him the membership. However, she needs to ask for a payment first, and therefore she initiates a new transaction 2. After the transaction 2 was initiated, the conditional rule was evaluated again. Now, the result of communication is not Undefined but False. This is because the T2 exists.

New transaction T02 was created with name=T02.2.

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:Request:Allow,T01.1:
State:Restrict

Step 4 – Elisabeth calculated a membership fee for Marek, and she requested a membership payment. The communication fact is still False.

Initiator of T02.2 performed Request.

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:Request:Allow,T01.1:
State:Restrict,T02.2:Request:Perform

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:Promise:Allow,T02.2:
Decline:Allow,T02.2:RevokeRequest:Allow,T01.1:State:Restrict

Step 5 – Marek promises to pay for the membership. Before he states the payment, he needs to fill the amount to pay based on the requested amount created by Elisabeth. He fills 30 Euro and states the payment. When transaction 2 is stated, a causal rule that validates if the paid amount is valid is activated. The sum of money matches and transaction 2 is accepted. Communication fact "Is membership paid?" is finally evaluated as True.

Executor of T02.2 performed Promise.

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:Promise:Allow,T02.2:
Decline:Allow,T02.2:RevokeRequest:Allow,T01.1:State:Restrict,T02.2:
Promise:Perform

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:State:Allow,T02.2:
RevokePromise:Allow,T01.1:State:Restrict

Executor of T02.2 performed State.

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:State:Allow,T02.2:
RevokePromise:Allow,T01.1:State:Restrict,T02.2:State:Perform

Fact "Is paid amount for membership enough?" was evaluated as True.

Fact "Is membership paid?" was evaluated as False.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:Accept:Allow,T02.2:Reject:Allow,T02.2:RevokeState:Allow,T02.2:Accept:Perform,T01.1:State:Restrict

Fact "Is membership paid?" was evaluated as True.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:RevokeAccept:Allow

Step 6 – Elisabeth is allowed to state the membership, and she does so. The communication fact “Is membership paid?” was evaluated once more because transaction 2 could have changed in the meantime.

Executor of T01.1 performed State.

Fact "Is membership paid?" was evaluated as True.

T01.1:State:Allow,T01.1:RevokePromise:Allow,T02.2:RevokeAccept:Allow,T01.1:State:Perform

T01.1:Accept:Allow,T01.1:Reject:Allow,T01.1:RevokeState:Allow,T02.2:RevokeAccept:Allow

Step 7 – Marek accepts the membership creation.

Initiator of T01.1 performed Accept.

T01.1:Accept:Allow,T01.1:Reject:Allow,T01.1:RevokeState:Allow,T02.2:RevokeAccept:Allow,T01.1:Accept:Perform

T01.1:RevokeAccept:Allow,T02.2:RevokeAccept:Allow

Step 8 - Marek is a proud member of Volley club. We can see that his record was created in the database. The *TransactionId* is there to associate the DEMO engine transaction instance identifier with the membership record. The relation could be also stored inside the DEMO engine as transaction instance’s external identifier.

6 Conclusions and Further Research

In this paper, we proposed a theoretical computation model called the DEMO Machine, and we demonstrated its capability to simulate DEMO models on a Volley club example. We strive to contribute to developing model-driven systems based on DEMO models. However, there are still many topics for further research. Apart from the specific topics mentioned in the text, we would like to stress the evolvability of DEMO models and its consequences, alignment with existing business process management systems, and adoption of DEMO-based systems for the end users, so they are easy to use and comprehend.

Acknowledgement. This research has been supported by CTU SGS grant No. SGS16/120/OHK3/1T/18.

References

1. Nuffel, D., Mulder, H., Kervel, S.: Enhancing the formal foundations of BPMN by enterprise ontology. In: Albani, A., Barjis, J., Dietz, J.L.G. (eds.) CIAO!/EOMAS -2009. LNBIP, vol. 34, pp. 115–129. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-01915-9_9](https://doi.org/10.1007/978-3-642-01915-9_9)

2. Skotnica, M., Kervel, S.J.H., Pergl, R.: Towards the ontological foundations for the software executable DEMO action and fact models. In: Aveiro, D., Pergl, R., Gouveia, D. (eds.) EEWc 2016. LNBIP, vol. 252, pp. 151–165. Springer, Cham (2016). doi:[10.1007/978-3-319-39567-8_10](https://doi.org/10.1007/978-3-319-39567-8_10)
3. Van Kervel, S.J.H.: Ontology driven enterprise information systems engineering. TU Delft, Delft University of Technology (2012)
4. Guizzardi, G.: Ontological foundations for structural conceptual models, vol. 015. University of Twente, Enschede (2005)
5. Dietz, J.L.G.: Enterprise Ontology Theory and Methodology. Springer, Heidelberg (2006)
6. Van Kervel, S., Dietz, J., Hintzen, J., Van Meeuwen, T., Zijlstra, B.: Enterprise ontology driven software engineering. In: Proceedings of the 7th International Conference on Software Paradigm Trends, ICSOFT 2012, pp. 205–210 (2012).
7. Figueira, C., Aveiro, D.: A new action rule syntax for DEMO MODEls based automatic worKflow procEss geneRation (DEMOBAKER). In: Aveiro, D., Tribolet, J., Gouveia, D. (eds.) EEWc 2014. LNBIP, vol. 174, pp. 46–60. Springer, Cham (2014). doi:[10.1007/978-3-319-06505-2_4](https://doi.org/10.1007/978-3-319-06505-2_4)
8. Huysmans, P., Oorts, G., Bruyn, P., Mannaert, H., Verelst, J.: Positioning the normalized systems theory in a design theory framework. In: Shishkov, B. (ed.) BMSD 2012. LNBIP, vol. 142, pp. 43–63. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-37478-4_3](https://doi.org/10.1007/978-3-642-37478-4_3)
9. Krouwel, M.R., Op ’t Land, M.: Combining DEMO and normalized systems for developing agile enterprise information systems. In: Albani, A., Dietz, J.L.G., Verelst, J. (eds.) EEWc 2011. LNBIP, vol. 79, pp. 31–45. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-21058-7_3](https://doi.org/10.1007/978-3-642-21058-7_3)
10. Op’t Land, M.: Exploring normalized systems potential for dutch MoD’s Agility (2011). Accessed 25 April 2014
11. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc. **s2-42**(1), 230–265 (1937)
12. Dietz, J.L.: The Essence of Organization - an Introduction to Enterprise Engineering. Sapio bv (2012)
13. Jan, D., Jan, H.: Theories in Enterprise Engineering Memorandum - TAO
14. Banks, J., Carson, J.S., Nelson, B.L., Nicol, D.M.: Discrete-Event System Simulation, 3rd edn. Prentice Hall, Upper Saddle River (2000)
15. Dietz, J.L.G.: Enterprise ontology - understanding the essence of organizational operation. In: Chen, C.S., Filipe, J., Seruca, I., Cordeiro, J. (eds.) Enterprise Information Systems VII, pp. 19–30. Springer, Dordrecht (2006)

Advances in Enterprise Engineering XI
7th Enterprise Engineering Working Conference, EEWC
2017, Antwerp, Belgium, May 8-12, 2017, Proceedings
Aveiro, D.; Pergl, R.; Guizzardi, G.; Almeida, J.P.;
Magalhães, R.; Lekkerkerk, H. (Eds.)
2017, XII, 235 p. 62 illus., Softcover
ISBN: 978-3-319-57954-2