DEMO based Dynamic Information System Modeller and Executer

Magno Andrade[1], David Aveiro[1,2] and Duarte Pinto[1]
[1]*Madeira Interactive Technologies Institute, Caminho da Penteada, 9020-105 Funchal, Portugal*
[2]*Faculty of Exact Sciences and Engineering, University of Madeira, Caminho da Penteada 9020-105 Funchal, Portugal*

Keywords: Enterprise Engineering, Workflow, EMEaaS, DISME.

Abstract: This paper presents a different approach to information systems named as Enterprise Modelling and Execution as a Service (EMEaaS). This approach, based on the DEMO methodology, has as objective to solve some of the issues found on other approaches such as software as a service or business processes as a service like the dependency from an outside third party to the organization. As a concrete implementation of this EMEaaS approach, we present a DEMO Based prototype called Dynamic Information System Modeller and Executer (DISME). DISME is a dynamic information system modeller that aims to serve at the same time as: (1) an organization modeller, (2) an information system and (3) a workflow management system that can be adapted to most organizational realities with no need for coding, just the understanding of some basics about the DEMO methodology.

1 INTRODUCTION

In modern days software is everywhere in our lives. The same concept applies to organizations, where to survive an ever more competitive market one needs tools to support the everyday processes and decision-making.

The field of Enterprise Engineering can help to create a strategic and global vision of the business and its specificities that, later, allow a systematic requirements elicitation and implementation that more effectively supports the management and information needs of an organization.

Size and complexity of organizations make it difficult to manage their processes hindering their efficiency and productivity. The information systems complexity grows hand in hand with the organization complexity increasing the challenges in properly capturing its essence and making it difficult for the information systems to fulfil the expectations on the organizational issues they were built to solve.

The particular need for the DEMO Based Dynamic Information System Modeller and Executer (DB DISME) derives from the lack of anything that is at the same time a DEMO based information system, a workflow manager and a modeller in a properly integrated fashion.

The development of this software tool and its conceptualization is, simultaneously, being tested with concrete processes of two organizations in different fields, one in cargo transportation and another being a government entity with their specificities in order to create a robust program capable of adapting to most organizational realities.

In the second section of the paper, we present the basic notions needed to understand the basic concepts and inner workings of the system as well as the need and motivation for DISME, the approach behind it and other tools that serve a similar objective. In the third section, we present the Dynamic Information System Modeller and Executer tool; explain how it works followed by a discussion and some remarks regarding current and future work on the fourth section. Lastly, on the fifth section we have the conclusions and main contributions.

2 CONTEXTUALIZATION

To a further understanding of the tool currently under development, we first need to understand some basics of the DEMO methodology. DEMO, allows one to specify a concise but comprehensive view of the organization, giving the perfect foundation to build the presented prototype.

The stability of its ontological models, highly abstracted from the human and technological means that implement and operate an organization also was

Andrade, M., Aveiro, D. and Pinto, D.
DEMO based Dynamic Information System Modeller and Executer.
DOI: 10.5220/0007230003830390
In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) - Volume 2: KEOD*, pages 383-390
ISBN: 978-989-758-330-8

also taken into consideration thinking of the other main part of this project that involves the development of a diagram editor to visually represent (using DEMO models) the organization and its processes.

Another basic notion is the type square pattern widely used in the developed database which allows the use of the system both as modeller of the organization's reality and it's respective production (and testing) information system.

Having the background notions, we also explain the reasoning behind the necessity of an Enterprise Modelling and Execution as a Service (EMEaaS) approach, and what makes it different from the multiple other approaches such as Automatic Code Generation, Software as a Service (SaaS) or Business Process as a Service (BPaaS).

Finally, in this section, we also present some related software and approaches that have similar functionalities of those implemented in DISME, and how our software differs from these tools.

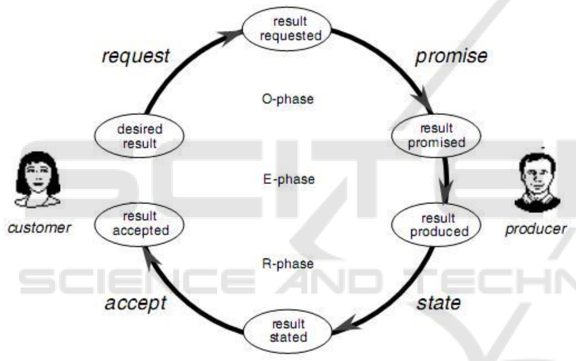


Figure 1: Actors Interaction with Production and Coordination Worlds.

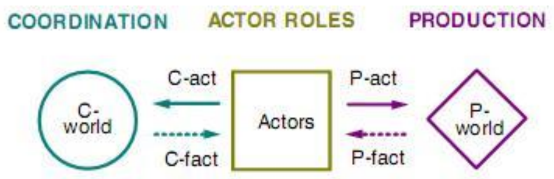


Figure 2: Basic Transaction Pattern.

2.1 DEMO - Operation, Transaction and Distinction Axioms

In the Ψ-theory (Dietz, 2009) – on which DEMO is based – the operation axiom (Dietz 2006) states that, in organizations, subjects perform two kinds of acts: production acts that have an effect in the production world or P-world and coordination acts that have an effect on the coordination world or C-world. Subjects are actors performing an actor role responsible for the execution of these acts. At any moment, these worlds are in a particular state specified by the C-facts and P-facts respectively occurred until that moment in time. When active, actors take the current state of the P-world and the C-world into account. C-facts serve as agenda for actors, which they constantly try to deal with. In other words, actors interact by means of creating and dealing with C-facts. This interaction between the actors and the worlds is illustrated in Figure 1. It depicts the operational principle of organizations where actors are committed to deal adequately with their agenda. The production acts contribute towards the organization's objectives by bringing about or delivering products and/or services to the organization's environment and coordination acts are the way actors enter into and comply with commitments towards achieving a certain production fact (Dietz, 2011).

According to the Ψ-theory's transaction axiom the coordination acts follow a certain path along a generic universal pattern called transaction (Dietz, 2006). The transaction pattern has three phases: (1) the order phase, were the initiating actor role of the transaction expresses his wishes in the shape of a request, and the executing actor role promises to produce the desired result; (2) the execution phase where the executing actor role produces in fact the desired result; and (3) the result phase, where the executing actor role states the produced result and the initiating actor role accepts that result, thus effectively concluding the transaction. This sequence is known as the basic transaction pattern, illustrated in Figure 2, and only considers the "happy case" where everything happens according to the expected outcomes. All these five mandatory steps must happen so that a new production fact is realized. In (Dietz, 2011). we find the universal transaction pattern that also considers many other coordination acts, including cancellations and rejections that may happen at every step of the "happy path".

Even though all transactions go through the four – social commitment – coordination acts of request, promise, state and accept, these may be performed tacitly, i.e. without any kind of explicit communication happening. This may happen due to the traditional "no news is good news" rule or pure forgetfulness, which can lead to severe business breakdown. Thus the importance of always considering the full transaction pattern and the initiator and executor roles when designing organizations (Dietz, 2011).

The distinction axiom from the Ψ-theory states that three human abilities play a significant role in an organization's operation: (1) the forma ability that concerns datalogical actions; (2) the informa that concerns infological actions; and (3) the performa that concerns ontological actions (Dietz, 2006). Regarding coordination acts, the performa ability may be considered the essential human ability for doing any kind of business as it concerns being able to engage into commitments either as a performer or as an addressee of a coordination act (Dietz, 2011). When it comes to production, the performa ability concerns the business actors. Those are the actors who perform production acts like deciding or judging or producing new and original (non derivable) things, thus realizing the organization's production facts. The informa ability on the other hand concerns the intellectual actors, the ones who perform infological acts like deriving or computing already existing facts. Finally, the forma ability concerns the datalogical actors, the ones who perform datalogical acts like gathering, distributing or storing documents and or data. The organization theorem states that actors in each of these abilities form three kinds of systems whereas the D-organization supports the I-organization with datalogical services and the I-organization supports the B-organization (from Business=Ontological) with informational services (Dietz and Albani, 2005). By applying these axioms, DEMO is claimed to be able to produce concise, coherent and complete models with a reduction of around 90% in complexity, compared to traditional approaches like flowcharts and BPMN (Dietz, 2008) (Aveiro and Pinto, 2013a).

2.2 Type Square Pattern

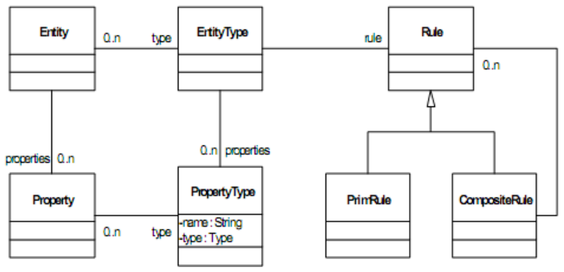


Figure 3: Type Square Pattern.

The type square pattern (Yoder et al., 2001), depicted in Figure 3, derives from the type-object pattern applied twice. The first time to separate the entities from their entity types and the second to separate attributes from their attribute types. This separation has as advantages: the possibility of creating new entity types and property types in run time, a great reduction in the number of subclasses - having multiple type instances instead – and the possibility to, in a dynamic fashion, change the type classes by changing the main class from where they derive from.

2.3 Automatic Code Generation

Automatic code generators, generate source code from a designed model, and can be very helpful tools depending on the task. There are some known advantages to it, like the time-saving it offers from not having to code everything from scratch, as long as the model was well specified (Kornecki and Johri, 2006).

However, there are also some shortcomings of using an automatic code generator, starting with the difficulty to maintain it. Because it usually generates an abstract solution that fits one or more problems, it may have many unnecessary code lines (for your needs) that you need to consider when doing any changes. This leads to another usual shortcoming, the low flexibility and the complexity in the customizations. These need to be very precisely introduced on the generator to fit one's particular needs, because the process of later editing might present itself as quite challenging (Etheredge, 2009).

There is also a high level of dependency on the code generator for any new version of the system, because of difficulties derived from the complexity of the code. The most cost-effective solution might be to generate the code again with different parameters rather than modifying the existing one.

Code generators are still very useful tools, and they are particular efficient when applied to something that hardly need change, menial coding tasks like writing pages which are nothing more than containers for data from the database.

2.4 Software as a Service

Software as a Service (SaaS) has a subscription service as its model, where the software is licensed from a proprietary organization and delivered based on a centralized host usually being accessed through the usage of a web browser (Gil, 2018).

The main reasons for the adoption of SaaS models are usual the implementation time and cost savings associated as well as being somehow easy to use since it is made for multiple customers. Other advantages are the scalability and accessibility. Because it caters to multiple organizations, it is likely that the solution one is looking for has been already implemented and can be acquired ready to be used. (Turco, 2013).

SaaS also has some disadvantages, for one its applications usually focuses on a specific field (e.g.

invoicing, CRM, etc). It is also not the most flexible solution, because it is rarely developed thinking about one organization in particular. The solution is generic, and some particularities of one's organization may not fit and therefore have to be dealt with in another fashion.

Besides these there are also some disadvantages derived from being a service accessed elsewhere such as response times, availability or, the most concerning, data security since an organization's information is being stored in the provider's server.

Other disadvantages include sometimes being forced to upgrade to a new version of the software or suffer from lack of support from the provider or even be faced with the provider going out of business and losing access (even if temporary) to the organization's data.

2.5 Business Process as a Service

Same as SaaS Business Process as a Service (BPaaS) is also a subscription model of software cloud based that automates the business process (task or set of tasks) of an organization designed in a way to be service oriented. The BPaaS is the top layer on top of the other cloud services, the System as a Service but also Platform as a Service as well as Infrastructure as a Service (Duipmans, 2012), , (Hurwitz et al., 2012).

These specific solutions, inherited most of the advantages from SaaS's such as the cost and time efficiency, scalability, accessibility, standardization or specialized staff on the server end dealing with every problem that may arise (Duipmans, 2012).

But, likewise, it also inherited most of the disadvantages from being dependent on an external provider like availability or data security concerns as well as lack of flexibility or an dependence of the remaining service stacks (SaaS, PaaS and IaaS) (Duipmans, 2012).

2.6 Enterprise Modelling and Execution as a Service

What we propose in this paper is a different approach, the Enterprise Modelling and Execution as a Service (EMEaaS).

Our vision of EMEaaS is that any worker can design enterprise processes based on DEMO language with no need for any specific programming knowledge.

These designed models can then be executed instantly after being designed.

We also aim for the inclusion of available pre-designed models (for specific businesses and/or other institutions) and fully customizable interfaces generated automatically based on model elements.

This service can be provided locally in an internal enterprise server or cloud based, thus solving one of the issues of the other approaches and cutting or heavily reducing the dependence of a service provider since all the process modelling and updating can be done "in house".

With these aims we propose our DISME vision and prototype as an EMEaaS solution.

2.7 Similar Platforms

If only considering the overall goal alone DISME can be compared with solutions such as Mendix ("Mendix Platform," 2018) and Appgyver ("AppGyver," 2018).

Mendix is an application that provides users with the power to build and continuously improve custom applications at an unprecedented scale and speed. The application gives the possibility to build mobile and internet applications. It provides a set of tools for the entire life cycle of an application ("Mendix Platform," 2018).

The AppGyver solution allows the creation of visually advanced logical applications; building business rules; automating technologies for the universe of email notifications and emails without the use of code; creating application interfaces with predefined libraries and with drag-and-drop components; defining and structuring page structures and navigation graphically. It also allows one to perform dynamic searches and combine multiple data sources ("AppGyver," 2018).

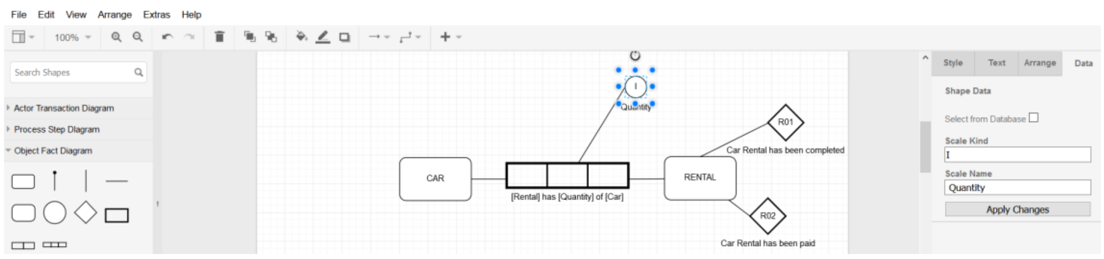
The two main features of Mendix and AppGyver are present in DISME but with two distinctions.

The first is that Mendix and AppGyver have a graphical drag-and-drop feature where you drop each element in the place where it is intended to be placed.

In DISME the properties of a form are inserted through a form, however the generation of these fields inserted in the control panel is generated automatically and dynamically as it happens in Mendix and AppGyver. However our DISME solution will soon include these drag-and-drop functionalities.

The second functionality is the construction of the application logic. A microflow / process can perform various actions such as creating and updating objects, presenting pages, making choices, and defining what employees have to do.

It is a visual way of expressing what is normally done in programming code. The notation used in the microflows is based on BPMN ("Microflows - Mendix Documentation," 2018, p. 7).



Shape Data
Select from Database ☐
Scale Kind: I
Scale Name: Quantity

Figure 4: OFD drawn using the DISME Diagram Editor.

The main difference between the platforms in this functionality is that DISME is governed by the DEMO modelling notation and every transaction flow of a process follows the rules described and specified by its modelling. The data flow and execution order of the transactions are a set of rules defined using the DEMO methodology.

Another related solution is the DEMO Processor (Van Kervel, 2012) but is limited in the sense that it focuses only on the coordiation aspects of transactions, leaving out the infological and datalogical actions to be implemented by external systems.

3 REALIZING THE EMEaaS VISION WITH DISME

The DISME solution has three main functionalities: 1) the Diagram Editor to create and view DEMO models 2) the System Modeller to adapt and parametrize in more detail the information system to the needs of the organization; and the System Executer that runs in production mode the modelled information system.

In the System Modeller, one of more users take upon their selves the administrator role, and are able to shape each process of an organization creating and editing transactions, their relations as well as associating input forms to these transactions, or in specific transactions steps. These forms are dinamically generated by the System Executer when users are fulfilling their organizational tasks The users that model the system, have no need for any specific programing skill only some basic knowledge of enterprise engineering modelling which is close to the "language / representation" used within organizations.

In the System Executer, users that have acquired permissions to take part in the transactions do so according to their roles following DEMO's transaction pattern.

The development of the database behind the DISME solution was heavily influenced by the DEMO way of thinking, trying to capture the essence of an organization's workflow, but without abstracting from their infological and datalogical implementations.

The goal was to keep the platform as flexible as possible in terms of the editing possibilities available.

3.1 Diagram Editor

The diagram editor of DISME was inspired in the Universal Enterprise Adaptive Object Model (Aveiro and Pinto, 2013b) and uses GraphEditor ("mxgraph," 2018) as a starting point. The main objective of that tool is to present the visual representation of the implemented diagrams in DISME while being a fully functional editor, but also to facilitate the design of new processes in a visual fashion and automate some of the steps in their implementation if so is desired.

This component is still in an early stage of development when concerning the interactions with the System Modeller of DISME. Still it's already a functional diagram editor with the ability to create DEMO models as shown in figure 4.

3.2 Conceptual Structure

In the following paragraphs, we have the object fact diagram (OFD) of DISME's conceptual model divided into its relevant parts. The darker object classes represent the tables responsible for storing the organization's processes specified in the DEMO diagrams whereas the light coloured object classes represent the tables that store instances of all types (processes, transactions, entities, values, etc.) as the System Executer is run in the day-to-day operation of the enterprise.

3.3 System Modeller

The System Modeller has many components, which are briefly explained next using examples from a car dealership for better understanding.

Users Management – creating new users, changing user data, or associating users with roles. Users are real individuals that interact with the system.
Roles Management – creating/editing roles, and associate them with users and actors. A role can fulfil various actors and an actor can be fulfilled by several roles. Roles are the traditional job title the individuals on the organization have, for example front desk clerk.

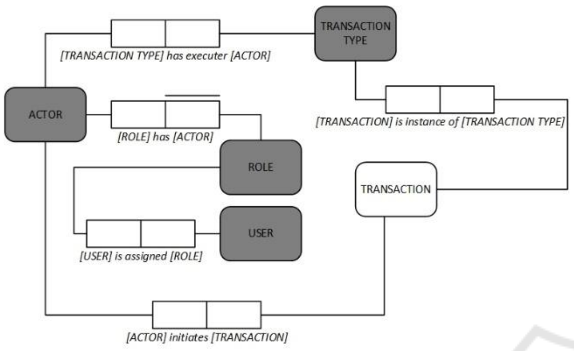

TRANSACTION TYPE
[TRANSACTION TYPE] has executer [ACTOR]
ACTOR
[TRANSACTION] is instance of [TRANSACTION TYPE]
[ROLE] has [ACTOR]
ROLE
TRANSACTION
USER
[USER] is assigned [ROLE]
[ACTOR] initiates [TRANSACTION]

Figure 5: Actors, Roles and Users - DISME OFD.

Actors Management – creating/editing actors and associating them with roles. An actor may be associated with several roles and these actors are responsible for starting and executing transactions. Actors are the DEMO actor roles, for example Renter that could in turn be portrayed by the front desk clerk Role.
Process Management – defining process types. These process types make up the set of processes that a particular organization covers in its business area. The purpose of the process type is to define the structure that the System Execution part of the platform uses to create the instances of processes that occur within a company. Car Rental could be an example of a process.

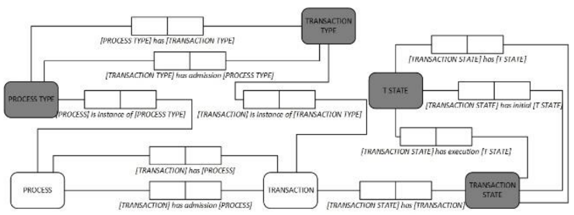

TRANSACTION TYPE
[PROCESS TYPE] has [TRANSACTION TYPE]
[TRANSACTION STATE] has [T STATE]
[TRANSACTION TYPE] has admission [PROCESS TYPE]
PROCESS TYPE
T STATE
[PROCESS] is instance of [PROCESS TYPE]
[TRANSACTION] is instance of [TRANSACTION TYPE]
[TRANSACTION STATE] has initial [T STATE]
[TRANSACTION STATE] has execution [T STATE]
[TRANSACTION] has [PROCESS]
PROCESS
TRANSACTION
TRANSACTION STATE
[TRANSACTION] has admission [PROCESS]
[TRANSACTION STATE] has [TRANSACTION]

Figure 6: Processes and Transactions - DISME OFD.

Transaction Management – This area is divided in two parts, the transaction types and the transaction acts.

The first part is for defining the types of transactions that are captured by applying the DEMO methodology. A transaction type is always associated with a process type and an executing actor. To a process type, because a transaction can only belong to one of the existing sets of process types so that the generated transaction can be associated with a process instance. And to an executing actor because following the DEMO paradigm models a transaction has one and only one executor who is responsible for the transaction acts standardized by the theories and models. For example a Car Delivery transaction type would belong to the Car Rental Process and executed by a Car Deliverer Actor.

The second part is for placing the transactions acts existing in the DEMO methodology that may require some action, in concrete, request, promise, execution, declaration and acceptance.

The transaction acts are extremely important because they are later used to define where relationship between various transactions take place as well as what needs to happen in the workflow to continue the process.

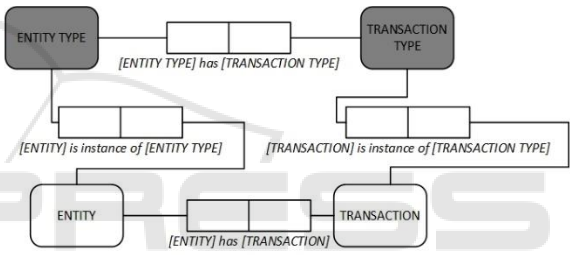

ENTITY TYPE
TRANSACTION TYPE
[ENTITY TYPE] has [TRANSACTION TYPE]
[ENTITY] is instance of [ENTITY TYPE]
[TRANSACTION] is instance of [TRANSACTION TYPE]
ENTITY
TRANSACTION
[ENTITY] has [TRANSACTION]

Figure 7: Entities and Transactions - DISME OFD.

Entities Management – Defining what could be compared to the definition of the table in a database that will be responsible for saving the corresponding records / data.

An entity type corresponds here to each OFD class that would exist in the organization model. Note that the entity type is nothing more than the structural definition of what should be stored in the database such as the functionality of a table. Car, would be an example of an Entity Type in this scenario.
Properties Management – Specifying, defining and associating property types to entity type or relationship type, namely specifying its name, value type (text, int, enum, etc.) and field type (to be output in the automatically generated forms of the interface), etc. A property type example for the Entity Type Car could be Licence Plate Number.
Allowed Value Management – Here happens the specification of values allowed for the properties of type ENUM. Car Manufacturer for example.
Relations Management – Defining many-to-many situations. The function of one type of relation is the

same as multiplicity in a database. Properties can also be associated to relationships.

Units Management – Denoting a certain unit of measure in unit symbol format, such as kg (kilograms), l (liters). For example horsepower in a Power property.

Causal Links – Connection between transactional acts that originate the beginning of other transactions dependent on the first automatically. With these rules, the users do not need to manually start the transactions that follows in the process flow. These rules after being set are automatically applied by the dashboard. For example, a Car Renting transaction type would have a causal link at the promise step to a Rental Payment transaction type.

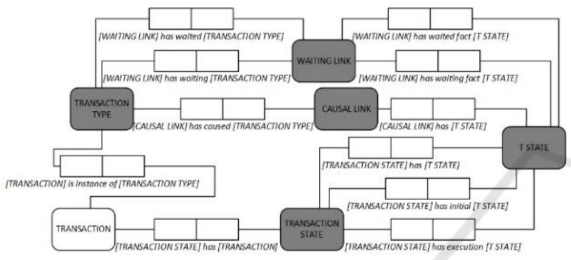

[WAITING LINK] has waited [TRANSACTION TYPE]
[WAITING LINK] has waited fact [T STATE]
WAITING LINK
[WAITING LINK] has waiting [TRANSACTION TYPE]
[WAITING LINK] has waiting fact [T STATE]
TRANSACTION TYPE
CAUSAL LINK
[CAUSAL LINK] has caused [TRANSACTION TYPE]
[CAUSAL LINK] has [T STATE]
T STATE
[TRANSACTION STATE] has [T STATE]
[TRANSACTION] is instance of [TRANSACTION TYPE]
[TRANSACTION STATE] has initial [T STATE]
TRANSACTION
TRANSACTION STATE
[TRANSACTION STATE] has [TRANSACTION]
[TRANSACTION STATE] has execution [T STATE]

Figure 8: Transactions and Links - DISME OFD.

Waiting Links – Connections between transactions that are only allowed to continue to the next defined act, if and only if the transaction act of another transaction have already been performed by the user. In these links the transaction that "waits" for the other necessarily needs information from that previous one to proceed. As an example the Car Renting transaction type would have a waiting link at the execution step where it would have to wait for the execution of the Rental Payment transaction Type.

3.4 System Execution

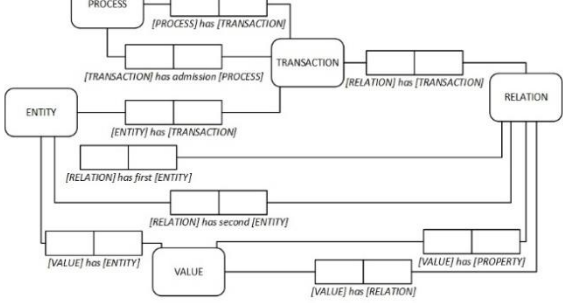

PROCESS
[PROCESS] has [TRANSACTION]
[TRANSACTION] has admission [PROCESS]
TRANSACTION
[RELATION] has [TRANSACTION]
RELATION
ENTITY
[ENTITY] has [TRANSACTION]
[RELATION] has first [ENTITY]
[RELATION] has second [ENTITY]
[VALUE] has [ENTITY]
VALUE
[VALUE] has [PROPERTY]
[VALUE] has [RELATION]

Figure 9: Working environment Classes - DISME OFD.

All users when logged in DISME are directed to the Dashboard where a list of the tasks they are allowed to perform is shown.

A user can execute a request act to start some specific process or react to a certain process state to which he or she were given authority and responsibility to do so – if some property/entity is associated to that act the user will have to fill out a form, automatically generated based on the specified parameters.

The Dashboard automatically controls the flow and state of all process instances and data, thanks to both the causal and waiting links that were specified in the respective modelling functions and the data submitted by the users.

Using the Car Rental example, a concrete customer paying, picking up and delivering a car would all be part of the System Execution creating instances of transactions of the defined transaction types and respecting the previously defined flow of events, this is, the parameters of the System Modeller.

4 DISCUSSION AND FUTURE WORK

DISME follows DEMO principles and as such, nothing is erased, an historic is kept of all changes in all concepts, both at type/model level and at runtime/instance level.

This leads to one of the points in the future work, the introduction of non-relational database. This has to be an alternative taken into consideration knowing the high number of relationships between tables and the costs of erasing nothing, probably even the implementation of a hybrid solution.

DISME's conceptual model follows in many parts the type square pattern and the principle of Adaptive Object Model and this is key to the ability of the system to immediately change its runtime behaviour according to the change in the specification of some concept at type/model level.

If a change is made at type/model level, it immediately reflects on the system's behaviour. For example, adding a new property to an existing entity type will result that the form generated in the respective transaction step will now show the respective field. For example, adding a weight property type to a driver Entity Type.

DISME currently only contemplates the four major types of DEMO acts; request, promise, execute, state and accept, in the future, there is the need to contemplate the full transaction pattern and include the cancelations, rejections, etc.

The interconnection of processes is another fundamental point to be implemented.

And also the forms behaviour that currently is mixed in the properties in the future should also lead to multiple interfaces for the same properties depending on the contexts.

5 CONCLUSION

The EMEaaS approach could solve some of the existing issues of the BPaaS most particularly the third party involvement and the derived security concerns. This is achieved offering as an alternative to outsourcing, the tools to facilitate the whole modelling process in house in a way that requires as little training and specific knowledge as possible.

Now this does not come without some trade-offs, although the tools are available, there is still the need for customization, and so the implementation times cannot be as fast as an out of the box solution, but still should be incomparably faster than developing an organization specific system.

Concerning DISME, it is an ongoing effort to validate the EMEaaS approach, and still has a long road to travel. Going down that path having partnerships with real large enterprises, as it is our case is invaluable as it is possible to foresee many problems that would not present themselves in a theoretical standpoint. This allied with the refinement over time, will lead to the production of a fully usable DISME prototype and large scale validation.

ACKNOWLEDGMENTS

This work was partially funded by FCT/MCTES LARSyS (UID/EEA/50009/2013 (2015-2018))

REFERENCES

AppGyver: Low-Code Enterprise-Grade App Creation [WWW Document], 2018. URL https://www.appgyver.com/ (accessed 3.12.18).

Aveiro, D., Pinto, D., 2013a. An e-Government Project Case Study: Validation of DEMO’s Qualities and Method/Tool Improvements, in: Harmsen, F., Proper, H.A. (Eds.), Practice-Driven Research on Enterprise Transformation, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, Utrecht, The Netherlands, pp. 1–15.

Aveiro, D., Pinto, D., 2013b. Universal Enterprise Adaptive Object Model. Presented at the 5th International Conference on Knowledge Engineering and Ontology Development (KEOD), Vilamoura, Portugal.

Dietz, J.L.G., 2011. Architecture – building strategy into design., in: Advances in Enterprise Engineering V. Springer.

Dietz, J.L.G., 2009. Is it PHI TAO PSI or Bullshit?, in: The Enterprise Engineering Series. Presented at the Methodologies for Enterprise Engineering symposium, TU Delft, Faculteit Elektrotechniek, Wiskunde en Informatica, Delft.

Dietz, J.L.G., 2008. On the Nature of Business Rules, in: Dietz, J.L.G., Albani, A., Barjis, J. (Eds.), Advances in Enterprise Engineering I, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 1–15.

Dietz, J.L.G., 2006. Enterprise Ontology: Theory and Methodology. Springer.

Dietz, J.L.G., Albani, A., 2005. Basic notions regarding business processes and supporting information systems. Requir. Eng. 10, 175–183. https://doi.org/10.1007/s00766-005-0002-9

Duipmans, E., 2012. Business process management in the cloud: business process as a service (BPaaS). University of Twente.

Etheredge, J., 2009. Code Generation Should be the Nuclear Option [WWW Document]. Simple Thread. URL https://www.simplethread.com/code-generation-should-be-the-nuclear-option/ (accessed 8.10.18).

Gil, P., 2018. “SaaS”: What Is “Software as a Service”? [WWW Document]. Lifewire. URL https://www.lifewire.com/what-is-saas-software-2483600 (accessed 3.28.18).

Hurwitz, J., Kaufman, M., Halper, F., Kirsch, D., 2012. What Is Business Process as a Service (BPaaS) in Cloud Computing? dummies.

Kornecki, A., Johri, S., 2006. Automatic Code Generation: Model-Code Semantic Consistency. pp. 191–197.

Mendix Platform [WWW Document], 2018. . Mendix. URL https://www.mendix.com/ (accessed 3.12.18).

Microflows - Mendix Documentation [WWW Document], 2018. URL https://docs.mendix.com/refguide/microflows (accessed 3.15.18).

mxgraph: mxGraph is a fully client side JavaScript diagramming library [WWW Document], 2018. URL https://github.com/jgraph/mxgraph (accessed 3.12.18).

Turco, K., 2013. Four Advantages of Software as a Service (SaaS) [WWW Document]. TechnologyAdvice. URL https://technologyadvice.com/blog/information-technology/four-advantages-of-software-as-a-service-saas-2/ (accessed 3.28.18).

Van Kervel, S.J.H., 2012. Ontology driven Enterprise Information Systems Engineering.

What is Business Process as a Service (BPaaS)? - Definition from Techopedia [WWW Document], 2018. . Techopedia.com. URL https://www.techopedia.com/definition/29543/business-process-as-a-service-bpaas (accessed 3.28.18).

Yoder, J.W., Balaguer, F., Johnson, R., 2001. Architecture and design of adaptive object-models. SIGPLAN Not 36, 50–60. https://doi.org/10.1145/583960.583966