Unified Service Model

Pedro Nuno Chaves Ornelas
Instituto Superior Técnico

October 21, 2009

Abstract

In the current economic context, being able to adapt our organizations is absolutely critical for remaining competitive or useful for the society at large. That adaptation is dependent on being able to reduce costs and increasing operational efficiency while simultaneously succeeding in innovating and staying relevant.

Enterprise Architecture promises to answer some of these challenges by integrating and aligning the organizational objectives with the reality of the systems that support them. Modeling reveals itself as a key element in design process of any Enterprise Architecture, by being responsible for the correct communication between the different stakeholders and therefore, critical on determining the success of these initiatives.

By making use of a convergent lexical and conceptual movement around the "service" concept, both in the social sciences and in the more technical areas, we propose the Unified Service Model (USM) modeling language. This language proposes to facilitate the communication between the different stakeholders and to minimize its own learning curve, by defining a set of shared concepts and symbols in a very frugal manner. We also propose a set of "a posteriori" analysis techniques, whose aim is to enrich the initial models with complementary information.

Finally, the proposed modeling language is analyzed and compared with relevant proposals in the same area and future improvements suggestions are made.

Keywords: Enterprise Architecture, Modeling, Service, Unified Service Model, USM.

1 Introduction

In an age where business models evolve into more intricate and interdependent relationships between several companies, the enterprise boundaries are increasingly expanding from the typical hierarchical organization to a network of organizations who depend on each other to survive. This increasing need for integration has created pressures in all organizational aspects, including the Information Technology (IT) function.

The integration and the alignment of business objectives with its information systems have been the main promises of Enterprise Architecture (EA) since it was introduced by John Zachman 22 years ago (Zachman, 1987), that describes and present and desired future state of the organization, and the process to transition between the two. But this transition is not exempt of difficulties: when crossing from one domain to another, and when stakeholders who hold information related to different areas are different, the communication is usually insufficient and becomes a heavy obstruction in developing a good solution (Lankhorst et al., 2009).

Even though EA has a considerably long history and a good adoption in the corporate world, executives are still considering increasing operational efficiency, with the development of new technologies or business opportunities has being simultaneously a priority and a desirable goal for their IT function (Roberts and Sikes, 2008). This leads us to believe that EA is still far from realizing its full potential and that significant improvements are still possible.

In an unrelated development, the convergence towards the "service" concept in the social sciences and in the IT fields has led us to question the potential utility of this concept, which we will call from now on the "service paradigm", as a communication bridge between different Enterprise stakeholders.

This work intends to capitalize on the service paradigm to create a modeling language which will enable a better communication during the EA process and thus, increasing the probability of its success.

2 Background

Architecture is a necessity in human endeavors like building a bridge or a microprocessor. To guarantee the coherence of the system being built, one must first rigorously define all the composing subsystems and their relationships. The Enterprise being a very complex system fully justifies the need for a similar solution, i.e., EA.

The *Enterprise*, being a very complex system, consisting of equally complex subsystems: the financial subsystem, IT subsystem, logistics, etc., can turn any attempt to fully grasp its state or to impose any future state, into a very daunting task. These subsystems, can in turn be designed and implemented in a myriad of different configurations, depending on the nature of the business, the organizational state and on the state of the environment.

The design decisions needed in an EA are made even harder by the simple existence of imperfect information across the various stakeholders, i.e., (almost) nobody has a complete picture of the structure and behavior of an entire *enterprise* nor have the necessary information needed to make some design decisions. EA must be able to communicate between the various stakeholders of an *Enterprise* to be successful in achieving the goals for whom it is intended, and thus, one needs to understand what are the factors that influence this communication effort. We argue that the main factors in a *successful communication* of a EA are:

a) Capability to choose proper modeling languages;
b) Capability to choose the proper modeling constructs;
c) Capability to set a shared vision and conceptual lexicon among stakeholders.

The modeling languages and constructs play a fundamental role in guaranteeing a good communication of the EA and hence, its success. But, what makes a modeling language proper for EA? In what way can it help the modeler in constructing a good model (i.e. choosing the proper modeling constructs)? And, can it help to create a shared vision and conceptual lexicon among stakeholders?

We argue that in order to to be able to achieve a good communication in a EA, a proper language must:

a) Strive to have the least possible amount of concepts, reducing its complexity and facilitating the learning process;
b) Use concepts which are shared among the biggest amount of stakeholders, increasing understanding and also facilitating the learning process;
c) Facilitate the design process by allowing the model to "evolve" in an iterative manner;
d) Permit the analysis and simulation of different aspects, allowing for a better understanding of the model and for better detection of errors in a earlier phase of a model's "evolution";
e) Allow for different stakeholders to understand how each others concerns are interrelated.

3 Related Work

The Open Group Architecture Framework (TOGAF) is one of the most used frameworks by organizations all over the world, and it is, at the time of this writing, at its ninth version (The Open Group, 2009b). This version of TOGAF has four mains pillars: (i) the *Architecture Development Method (ADM)*; (ii) the *Architecture Content Framework*; (iii) the *Enterprise Continuum* and the *Architecture Repository*; and (iv) the *Architecture Capability Framework*. These pillars will be described in the remaining of this section. TOGAF deals with four different kinds of architectures: (a) *Business Architecture*; (b) *Data Architecture*; (c) *Application Architecture*; and (d) *Technology Architecture*. They are constructed during the execution of the ADM which is a stepwise cyclical approach for the development of the overall enterprise architecture.

The *Zachman Framework* has a very historical significance in the field of EA, since it was the first framework to address areas beyond information systems architecture, such as the strategic and business operations areas. It is essentially a "six-by-six matrix with the communication interrogatives [*What*, *How*, *When*, *Who*, *Where*, and *Why*] as columns and the reification [*Identification*, *Definition*, *Representation*, *Specification*, *Configuration* and *Instantiation*] transformations as rows" (Zachman, 2008).

ArchiMate (The Open Group, 2009a) is the result of a two year project between several Dutch partners, and it is being hosted as a standard by the Open Group since 2008. It consists of a language framework which defines three layers into which languages constructs are positioned: (1) the *Business Layer*; (2) the *Application Layer*; and (3) the *Technology Layer*. A *metamodel* is defined which describes the language's basic entities which can establish different types of relationships between one another. Eighteen viewpoints are defined which show different perspectives of the same reality and quantitative and functional analysis techniques are defined which allow for a better understanding of the modeled reality.

Design and Engineering Methodology for Organizations (DEMO) (Dietz, 2006) is a theory and methodology based on *Language-Action Perspective (LAP)* (Weigand, 2006) and deeply rooted on *Speech-Act Theory* by Searle (1969), the main idea being that "by saying something, we do something", i.e., language is associated with some kind of action. DEMO defines a modeling method which is based on four aspect models: (1) the *Construction Model*; (2) the *Process Model*; (3) the *Action Model*; and (4) the *State Model*. Ettema and Dietz (2009), praise DEMO for coherence, consistency, comprehensiveness and conciseness due to its use of $\Psi - theory$.

E^3-value (Gordijn, 2002b,a) is a modeling language and methodology aimed at the requirement engineering of business ideas[1] and oriented towards the more business related stakeholders, i.e. CxOs and business analysts (Gordijn, 2003). The methodology provides three basic viewpoints, which represent increasingly levels of detail: a) Global actor viewpoint; b) Detailed actor viewpoint; and c) Value activity viewpoint. Each of this viewpoints are associated with a conceptual model, where the more detailed viewpoints extended from the simpler ones. The usage of the produced models is enriched with a "use case map", which is essentially a visual representation of some possible behavioral relations between the actors represented by the value models.

[1]It was developed with a special focus on e-commerce ideas.

Unified Modeling Language (UML) is Object Management Group (OMG)'s widespread "all-purpose", methodology independent, modeling language. With its origins in software systems development, UML as been gaining acceptance for modeling other kinds of systems, e.g., business process modeling (Eriksson and Penker, 2000). Extensibility mechanisms are available which allow the addition of new features in UML, which will then be grouped in a *Profile*. The model produced can be either platform-independent (PIM) or platform-specific (PSM), as one chooses. Service-Oriented Architecture Modeling Language (SoaML) is an under development profile for UML which aims at providing a standard way to model and architect Service-Oriented Architecture (SOA)s.

Business Process Modelling Notation (BPMN) is a graphical representation for specifying business processes in a workflow. The primary goal of BPMN is to provide a standard notation that is readily understandable by all business stakeholders, e.g., business analysts, technical developers, business managers, etc. It is intended to serve as a common language to bridge the communication gap that frequently occurs between business process design and implementation, being used therefore to capture business processes in the early phases of systems development (Dijkman et al., 2008).

Our proposal is based on the work by Delgado (2008) who proposes a model which is based on two paradigms: the object paradigm and the service paradigm, being the latter an extension of the former.

4 The Unified Service Model

We this proposal, we intend to create a modeling language which is common to most of the EA stakeholders. By guaranteeing that a common lexicon and conceptual model is shared, the communication between these people can be greatly improved. Naturally, this language should be useful to these stakeholders by addressing their different modeling needs while remaining conceptually simple and concise. This is what we mean by a "unified" modeling language.

As the name of this article implies, we used the "Service" concept to provide a shared lexicon, i.e., the service paradigm. We borrowed terms from different fields of study (Ornelas, 2009) to diminish the learning curve for most of the stakeholders, since they will already be familiar with some of these terms.

The higher-level architecture of the Unified Service Model (USM) is based on the *Principal Model*, *View Model* and *Visualization Model* technique, but in our proposal we divide the Principal Model into two separate models: the *Core Model* and the *Extended Model.* The *Principal Model* represents a coherent and complete representation of the domain being analyzed. The *View Model* will select some of the aspects from the Principal Model, so that they are relevant to the stakeholders in the context in which it is being communicated or manipulated. The *Visualization Model* is the symbolic model, and is composed of all the graphical symbols.

5 The Principal Model

We will now describe the core elements of the language. These elements are "axiomatic" propositions, since they are not derivable from any other elements, and will serve as the basis for defining other language elements in the Extended Model. A model built using only this set of concepts is called a *Core Model.* The elements are briefly defined as follows:

- *Service*: the application of competence for the benefit of another;
- *Service System*: any entity that is part of a service, i.e., an entity capable of applying some type of competence for the benefit of another;
- *Value Object*: an applied competence;
- *Value Port*: auxiliary concept which maps the exterior to the interior of the Service System;
- *Value Transfer*: the application of a competence from one SS to another;
- *Activity*: any kind of action performed by a Service System;
- *Event*: an observable occurrence which representing some kind of state change in the SS where it occurs;
- *Capability*: any kind of ability that a Service System may use when needed;
- *Sequence*: any list of ordered steps and is composed of three types of elements: events, capabilities and activities;

Both the *Activity* and the *Value Object* can be of a *continuous* or *discrete* nature, which has a significant impact on the modeled system.

The *Extended Model* is, as the name implies, an extension of the *Core Model* and does not share a *model* $\leftrightarrow$ *metamodel* relationship with it. Its elements are defined as follows:

- *Interface*: The *Interface* is an auxiliary concept which aims at grouping the *Value Ports.*
- *Transaction*: represents two or more *Value Transfers* between two *Service Systems* and using the same *Interface* in each of them.
- *Call & Wait Activity*: provides a "shortcut" which is analogous to a synchronous Remote Procedure Call.

Composition is allowed in both the structural dimension as in the behavioral dimension, but due to space limitations we will not be addressing them in detail.

6 Model Design

The USM defines three viewpoints which aim at allowing the direct manipulation of the Principal Model. As such, the elements (symbols with meaning) that we propose have an almost complete overlap with the concepts previously introduced. The elements that don't overlap are only used as visual aids and therefore are not converted from and to the Principal Model.

Visual consistency is important between different Viewpoints for guaranteeing a coherent interpretation of the symbols. We define a consistent set of graphical symbols (see Fig. 1).

The *Ecosystem Viewpoint* is the highest-level description of the proposed viewpoints. It shows the interactions between a set of Service Systems and abstracts completely from the internal behavior and structure from each of the Service Systems. An example of this viewpoint is given in Fig. 2

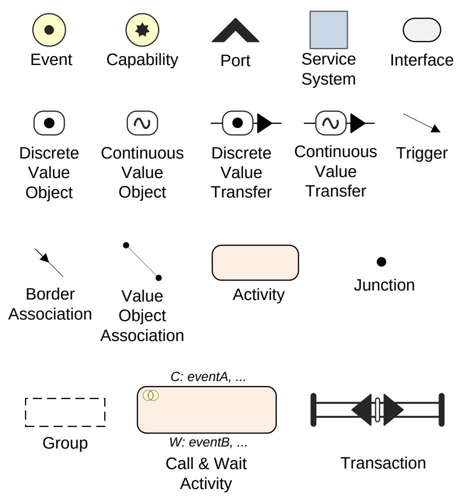

Figure 1: USM Symbols.

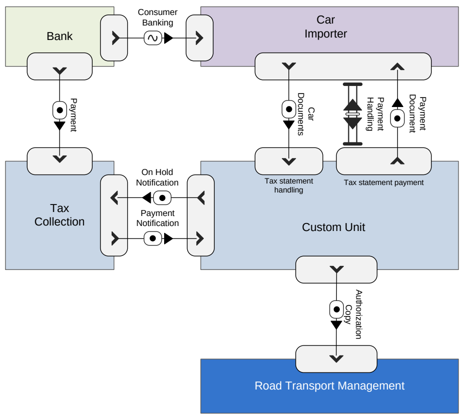

Figure 2: Ecosystem Viewpoint Example.

The *Border Viewpoint* represents the mapping from the exterior of a Service System to its interior. It shows the associations between the Ports and the Events or the Ports and the Capabilities.

The *Internal Viewpoint* is the representation of the internal behavior of a single Service System. One view created by this viewpoint represents all or part of the Sequences of that Service System. An example of this viewpoint is given in Fig. 3.

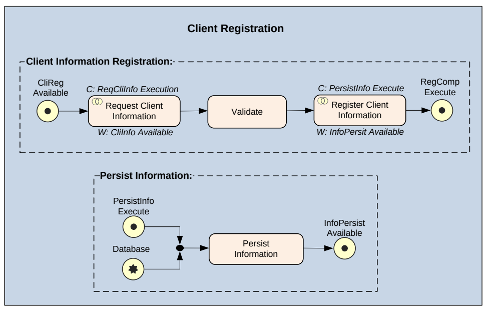

Figure 3: Internal Viewpoint Example.

7 Model Analysis

In order to better inform and complement the model, we describe two analysis techniques. The previously defined viewpoints allowed the stakeholder to visualize and simultaneously manipulate the *Principal Model*. This was possible due to a sequence of mapping activities that allowed the symbols to have an "isomorphism" with the *Extended Model* and vice-versa. However, the concepts that we will now present have an unidirectional mapping relationship to the Principal Model: they can only be mapped from a *Core Model*, but never to it.

These techniques are based on a dependency graph which is based on the model generated by the Viewpoints that were defined previously. We propose an algorithm for generating that graph below[2].

This is the main procedure which is executed when one wants to create a dependency graph, with starting element e:

```
CDG($e$)
1  CGE($e$)
2  if $e$ is a Port
3      then AP($e$)
4  elseif $e$ is Service System
5      then ASS($e$)
6      else  ABE($e$)
7  return graph
```

This procedure analyses a Service System element ss and constructs the dependency graph:

[2]Due to format limitations we had to shorten the algorithm references names. For a better description, please refer to the original essay.

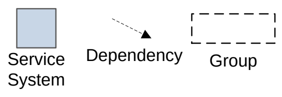

Figure 4: Dependency Viewpoint Symbols

```
ASS($ss$)
1  for each $e \in \text{BEs}(ss)$
2       do if $\text{Dep}(e) \in \emptyset$
3            then $\text{AGD}(ss, e)$
4                 $\text{ABE}(e)$
5  for each $p \in \text{IP}(ss)$
6       do $\text{AGD}(ss, p)$
7          $\text{AP}(p)$
```

This procedure analyses a Port element p and constructs the dependency graph:

```
AP($p$)
 1  if $p$ is an Input Port
 2    then
 3         if $\text{IEs}(p) \in \emptyset$
 4           then $ss \leftarrow \text{SSs}(p)$
 5                $\text{AGD}(p, ss)$
 6                $\text{ASS}(ss)$
 7           else for each $e \in \text{IEs}(p)$
 8                     do $\text{AGD}(p, e)$
 9                        $\text{ABE}(e)$
10    else  ▷ $p$ is an Output Port
11         for each $vt \in \text{VTs}(p)$
12              do $t \leftarrow \text{TP}(vt)$
13                 $\text{AGD}(p, t)$
14                 $\text{AP}(t)$
```

This procedure analyses a behavioral element e and constructs the dependency graph.

```
ABE($e$)
1  if $\text{Dep}(e) \in \emptyset$
2    then $ss \leftarrow \text{SSs}(e)$
3         $\text{AGD}(e, ss)$
4         $\text{ASS}(ss)$
5    else for each $d \in \text{Dep}(e)$
6              do $\text{AGD}(d, e)$
7                 if $d$ is a Port
8                   then $\text{AP}(d)$
9                   else $\text{ABE}(d)$
```

For someone who is designing a system, information about the dependencies between the various elements of that system is very useful to understand what the potential consequences of a change are in that system. The first technique proposes to supply the user with that information by being part of what we call the *Dependency Viewpoint*, which defines three symbols (see Fig. 4) and an algorithm which is based on the dependency graph we mentioned before. The algorithm is defined bellow.

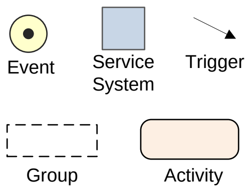

Figure 5: Sequence Viewpoint Symbols

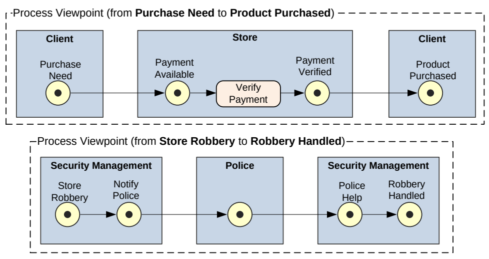

Figure 6: Sequence Viewpoint Example

This procedure is executed whenever one needs to build a Dependency viewpoint and its arguments are: the Service System s to which one wants to analyze its dependencies; and the set of Service Systems S that we wish to consider for analysis.

```
Create-DV($s, S$)
1  $\text{DSS}(s)$
2  $g \leftarrow \text{Create-DG}(s)$
3  for each $d \in \text{Interacting-SSs}(g, S)$
4       do $\text{DSS}(d)$
5          $\text{Link-Dependency-To}(s, d)$
```

These procedures analyses the graph and search for the Service Systems dependencies:

```
Interacting-SSs($s, S$)
1  $r \leftarrow \emptyset$
2  for each $d \in \text{Dependents}(s)$
3       do $r \leftarrow r \cap \text{ISS-Visit}(d, S)$
4  return $r$
```

	What	Who	How	When	Where	Why
ArchiMate	√	√	√	√	–	√
DEMO	√	√	√	√	–	–
USM	√	√	√	√	–	–
E^3-value	√	√	√	√	–	–
UML/SoaML	√	√	√	√	–	–
BPMN	–	–	√	√	–	–

Table 1: Modeling languages comparison of their primitive interrogatives support.

```
ISS-VISIT($g, S$)
1  if $g \in S$
2     then $r \leftarrow \{g\}$
3     else $r \leftarrow \emptyset$
4  for each $d \in$ DEPENDENTS($g$)
5        do $r \leftarrow r \cap$ ISS-VISIT($d, S$)
6  return $r$
```

A common approach to modeling consists on designing the behavior of a system in a first phase, and at a later stage design the structural aspects according to the previously designed behavior. Process modeling languages are commonly used for the graphical modeling of behavior, specially when the Enterprise is the system being modeled. The purpose of the viewpoint that we will now describe, is to provide a somewhat basic version of those modeling languages, based on the models that were designed with the viewpoints discussed in the previous section. It is intended to provide the stakeholder with a view of a sequence that would otherwise be modeled in distinct subsequences. This viewpoint defined five symbols (see Fig. 5) and an algorithm which is based on the dependency graph we mentioned before. The algorithm is defined bellow.

To create the viewpoint we will therefore need to know what are the "start" and "end" Events, as shown by the s and e arguments of CREATE-SV, which is used to build the Sequence viewpoint. This procedure only responsibility is to create the dependency graph and execute the CSV-VISIT procedure. An example of the outcome is depicted in Fig. 6.

```
CREATE-SV($s, e$)
   CSV-VISIT(CREATE-DG($e$), $s$)
```

The CSV-VISIT procedure, who is responsible for creating most of the viewpoint, can be seen as having two sequential phases: (1) search the graph; and (2) draw the viewpoint.

```
CSV-VISIT($g, s$)
1  if $g = s$
2     then DSS(SS($s$))
3          DRAW-BE($s$)
4          return $s$
5  for each $d \in$ DEPENDENTS($g$)
6        do $r \leftarrow$ CSV-VISIT($d, s$)
7           if $r \neq$ NIL
8              then if $g$ is a DA or an Event
9                      then $ssg \leftarrow$ SS($g$)
10                          if SS($r$) $\neq ssg$
11                             then DSS($ssg$)
12                          DRAW-BE($g$)
13                          DRAW-TT($r, g$)
14                 return $g$
15 return NIL
```

8 Discussion

We compared USM with other modeling languages according to the different interrogatives of the Zachman framework and we did not found big differences in the general support for each of those interrogatives. However, smaller details exist and we instruct the reader to the original essay for that discussion. A summary of those differences are found in Table 1.

A modeling language must strive to reduce its language footprint to a minimum, i.e., strive to have the least possible amount of concepts. As a direct consequence of this effort the complexity of the language is reduced and the learning curve is smoothened. With only seventeen graphic symbols, USM stands only behind of DEMO in terms of symbolic frugality amongst the languages in Section 3. Another important factor in influencing the steepness of the learning curve of any modeling language is the familiarity of the stakeholders with its concepts. With USM we made a conscious effort to mix both the business related concepts with some others from the IT related fields of study.

With some rigorous definitions, e.g., ORM diagrams, and other more textual and illustrative, USM provides a semi-formal semantics which is powerful and rigorous enough to allow the creation of algorithms to automatically analyze the constructed models, while providing a rigorous definition of the relationships between those concepts thereby minimizing the chance of mis-utilization of those concepts during the modeling effort.

USM was designed to achieve a balance between three different needs that are usually incompatible, due to their tendency to push the language in one direction or the other, i.e., towards a higher level language (idealization) or towards a more specific language (realization). These needs are described bellow:

1. To be detailed and precise in our model definitions, allowing for the analysis mentioned in Section 7;
2. To have a reusable modeling language across several Enterprise sub-domains (a unified modeling language);
3. To have a compact language which allows for a better learning process.

In the original essay a more detailed discussion is available along with a practical comparison between USM, ArchiMate and DEMO.

8.1 Limitations

While some of the discussed tools are attached to specific methodologies or views about the Enterprise, USM strives to be completely independent of any particular theory. But, it is rooted on the service paradigm, which supplies a specific lexicon and set of concepts. These concepts are focused more on the interrelationships between structural entities and on the behavior that leads to those interactions. As a consequence, other aspects are given less importance.

Another drawback of USM are the large number of generated diagrams, specially when compared to other languages, which may be a limiting factor for two main reasons: (1) time costs related of modeling and interpreting a large number of diagrams; (2) bad spatial economy, limiting the amount of information visualized simultaneously.

8.2 Comparisons

As a standalone tool, USM provides enough detail for much of the initial and middle phases of an EA process by providing the users with a general, yet rigorous overview of the enterprise (scope) being modeled. USM has a natural advantage over very conceptually complete languages like ArchiMate by supplying a simpler and more concise “language footprint” while being able to represent almost the same reality[3].

On the other hand, DEMO and e^3-value , approach the subject from a more limited set of concepts, which makes them ideal for the more premature and initial phases of an EA process while being unsuitable for representing details which are needed in later phases. Although USM is capable of representing the same reality than DEMO and e^3-value , these characteristics make them good complimentary languages, if used in different phases of the EA process.

UML stands apart of the discussed tools by being a more general modeling technique, even tough very complete, its various diagrams are somewhat detached of each other, making it more difficult to specify analysis techniques like the ones presented in this article and to form a coherent whole. This makes UML more adequate for use on smaller and more focused domain where the alignment and coherence aspects are easier and/or not so important to achieve. SoaML by being a profile, i.e., an extension of UML, has identical characteristics.

BPMN on the other hand is focused on representing the behavioral aspect of the Enterprise, making it insufficient for representing the same reality as USM, positioning them in a best case scenario as complimentary languages, and only due to BPMNs better and more complete grasp on the behavioral aspect of the Enterprise which may be required in some contexts.

9 Conclusions

We proposed a new modeling language which we called Unified Service Model (USM). This language is based on the “service paradigm”, i.e., a set of concepts and lexicon centered on the “service” concept. The language is constituted of a small set of seventeen symbols and three viewpoints, which are intended to be used by most of the stakeholders of an Enterprise Architecture (EA).

[3]With the exception of the “Why” aspect.

USM can position itself as both an alternative or as a complement to the current modeling tools by providing a relatively "light-weight" set of concepts and symbols that can be used by stakeholder in many of EA roles while being rigorous enough to allow "a posteriori" analysis to take place and enrich the initial model with complementary aspects.

We conclude that our proposed modeling language has therefore the potential do be used in an EA context to enhance communication between the different shareholders, thus enhancing the EA process itself.

9.1 Future Work

The use of automated analysis, like the ones proposed in this article, can greatly enhance the capabilities of a modeling language when used in conjunction with an automated (software) tool. By reducing the analysis processing to a negligible time, one can use the information provided by these analysis to change the initial model, in an iterative fashion.

These tools could also be a good solution to one of USM main problems, the large amount of diagrams. Much of the produced diagrams are "border viewpoints" which, although necessary to guarantee a coherent and complete model, do not always present the user with enough relevant information to make them useful in a practical context. A tool that could, through some kind of "intelligent algorithm", e.g. heuristic based algorithm, automatically create and fill these viewpoints by taking advantage of the information provided in the "ecosystem" and "internal" viewpoints, would greatly reduced the time penalty incurred by this limitation.

The addition of supporting concepts for the "Why" aspect of the organization may be an added benefit for the future development of the modeling language, allowing a better communication between the stakeholders who would better understand some design decisions.

Another future challenge for USM will be to be able to produce more concise diagrams, i.e. to: (1) generate diagrams which do not consume such a large amount of physical space; and (2) generate diagrams who can hold a larger amount of conceptual information. The first challenge can be overcome by tweaking the graphical symbols until a more compact form is achieved, thereby reducing the physical space it consumes. One must also be aware of the risk of cluttering, by having to much graphical detail, which will confuse instead of providing more information. The second challenge is better handled by extending the *Extended Model* with new concepts which will facilitate the understandability of the language by its users. Adding new concepts should be done with maximum parsimony, since having a small language footprint is an added benefit. A third option that could potentially address both of the issues would be the creation of one or more viewpoints that could not only better handle the distribution of graphical symbols, but also incorporate new concepts together.

References

Delgado, J. (2008). Modelação de sistemas com serviços unificados. Private notes.

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

Dijkman, R., Dumas, M., and Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294.

Eriksson, H. and Penker, M. (2000). *Business modeling with UML*. Wiley.

Ettema, R. and Dietz, J. (2009). ArchiMate and DEMO–Mates to Date?

Gordijn, J. (2002a). E3value in a nutshell. Technical report, HEC University Lausanne.

Gordijn, J. (2002b). *Value-based requirements Engineering: Exploring innovative e-commerce ideas*. PhD thesis, Vrije Universiteit Amsterdam.

Gordijn, J. (2003). Why visualization of e-business models matters. In *Proceedings of 16th Bled conference 2003*, pages 12–15.

Lankhorst, M., Proper, H., and Jonkers, H. (2009). The Architecture of the ArchiMate Language. In *Enterprise, Business-Process and Information Systems Modeling: 10th International Workshop, Bpmds 2009, and 14th International Conference, Emmsad 2009, Held at Caise 2009, Amsterdam, the Netherlands, June 8-9, 2009, Proceedings*, page 367. Springer.

Ornelas, P. (2009). Projecto de Mestrado em Eng Informática e Computadores. University Project.

Roberts, R. and Sikes, J. (2008). IT's Unmet Potential. *McKinsey Quaterly*.

Searle, J. (1969). *Speech acts: An essay in the philosophy of language*. Cambridge University Press.

The Open Group (2009a). *ArchiMate® 1.0 Specification*. Van Haren Publishing.

The Open Group (2009b). *TOGAF Version 9*. Van Haren Publishing.

Weigand, H. (2006). Two decades of the language-action perspective. *Communications of the ACM*, 49(5):45–46.

Zachman, J. (1987). A framework for information systems architecture. *IBM systems journal*.

Zachman, J. A. (2008). The Zachman Framework: The Official Concise Definition. Technical report, Zachman International.