Multi-level System Modeling Using the Foundation Concepts of RM-ODP

Alain Wegmann, Lam-Son Lê
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland
{Alain.Wegmann, LamSon.Le}@epfl.ch

Bryan Wood
Open IT
11 Wilton Court, Sheen Road
Richmond, TW9 1AH, UK
Bryan.Wood@Open-It.co.uk

Abstract

A specification in Enterprise Architecture (EA) requires the modeling of an enterprise across multiple levels, from the markets in which it operates down to the implementation of the IT systems that support its operations. Our goal is the development of a method and of a CAD tool that support such modeling.

To achieve our goal, we need an ontology to represent systematically all the systems at the multiple levels identified in an enterprise. We base our ontology on the foundation modeling concepts defined in Part 2 of ISO/ITU Standard "Reference Model of Open Distributed Processing" (RM-ODP). In this paper, we present how multi-level systems can be represented using directly the concepts defined in Part 2 of the RM-ODP.

Our modeling approach differs from that defined in Part 3 of the RM-ODP, which focuses on the specification of IT systems in terms of viewpoint models representing the IT system environment and its construction. The benefit of our approach is the capability to model systematically and consistently the multiple systems represented in a company.

Keywords: RM-ODP, Enterprise Architecture, Ontology, Multi-level Modeling, System Modeling.

1. Introduction

IT and business alignment is one of the top-ranked issues for Chief Information Officers (CIO) [1]. Enterprise architecture (EA) addresses this alignment issue. EA deals with the specification and design of systems that span from business entities (market, value network, business, department, employee…) down to IT entities (e.g. IT systems, applications, software components, programming language classes). Our goal is the development of an EA design method called *SEAM in EA* (*Systemic Enterprise Architecture Methodology in Enterprise Architecture*) [2] and of the corresponding tools [3]. When using SEAM in EA, the EA team develops an enterprise model that represents the company's environment, the company's roles, the company's organization, the IT system functionality and its construction. The enterprise model is represented with a notation similar to UML [4]. The model is defined by an ontology that makes systemic concepts (such as contexts, boundaries, etc…) explicit and that represents systematically all levels. This ontology is based on the foundation modeling concepts defined in Part 2 of ISO/ITU Standard *Reference Model of Open Distributed Processing (RM-ODP)* [5]. This paper discusses the applicability of the RM-ODP Part 2 concepts for this purpose.

Section 2 presents an example of multi-level system modeling using SEAM in EA. Section 3 discusses the applicability of RM-ODP Part 2 to multi-level system modeling. Section 4 outlines related work. Section 5 discusses the applicability of the proposed approach and outlines the future research directions.

2. Multi-Level Modeling: An Example

This section presents an example of multi-level modeling. The example illustrates the SEAM CAD tool (Section 2.1), the SEAM in EA notation and terminology (Section 2.2) and how to achieve traceability across functional levels (Section 2.3) and organizational levels (Section 2.4).

The example represents an `EA Project` in which there is a `ProductMarket`. This `ProductMarket` is composed of a `Supplier Value Network` (`SVN`) that

serves a Customer. *Value network* is a business term that is used to describe a group of companies that collaborate (to create value for a customer). The SVN is composed of three companies: MarketingCo, ManufacturingCo and ShippingCo. The company MarketingCo is composed of departments: WarehouseDep and PurchasingDep. In these departments, there are software applications and employees: OpApp and Clerk. This organizational breakdown can continue untill all relevant elements (possibly, for the software applications, down to the programming classes) are identified and designed. In SEAM, we consider all the elements enumerated above as entities that are perceived as systems. Figure 1 represents informally this set of systems. In this paper we focus on the upper three levels.

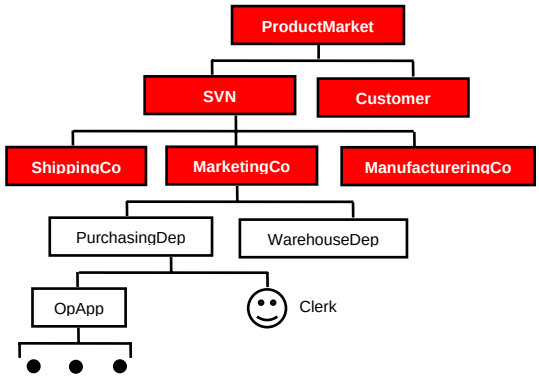

Figure 1: Informal description of ProductMarket

In this Section we present four representations of the SVN. The SVN is considered as the *system of interest* in our example. The first one (Section 2.1) represents the SVN as a whole at the highest functional level (i.e. only one SVN action is visible: the action Lifecycle). The SVN is represented interacting with a Customer. The SVN and the Customer are components of the ProductMarket. The second representation (Section 2.2) represents SVN at a more detailed functional level. The SVN's Lifecycle becomes an activity that includes the SVN's initialization, one Sell action and the SVN's termination. The SVN's Sell action corresponds to the SVN's role in the interaction Sale that takes place between the system of interest and the Customer within the ProductMarket. In the third representation (Section 2.3), the Sell action of the system of interest is detailed: the customer selects a product in SVN, the customer's payment info is verified by SVN and the product is delivered to the Customer by SVN. The fourth and last representation (Section 4) describes the system of interest at a different organizational level. It presents the SVN as a composite: a group of three companies that collaborate to perform the sell&deliver action. This action is the implementation of the Sell action or activity described in the previous representations. The role of each company is analyzed.

These four representations illustrate the notions of multi-level modeling.

2.1. CAD Tool Overview

In SEAM, we represent multiple systems. One of the challenges for the modelers is to understand and manage all the diagrams that represent the different aspects of these systems. For this purpose we have developed a notation [6] that makes contextual information explicit and a CAD tool [3] that manages the model. In this paper, all SEAM diagrams are snapshots taken from the CAD tool.

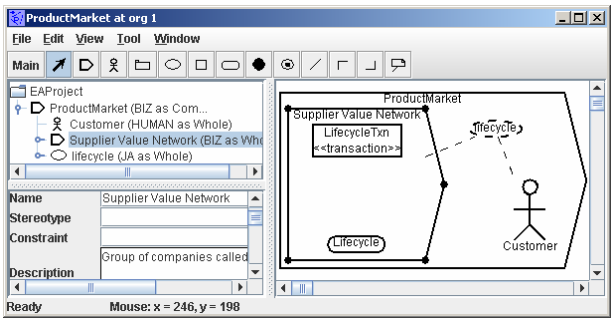

Figure 2: Snapshot of the CAD tool [3] representing ProductMarket composed of SVN and Customer.

Figure 2 is an editing window. To manage different parts of the same enterprise model, multiple editing windows can be opened at the same time. The window has three panels: The diagram panel (right side of the window) presents graphically a selected view of the model. The tree panel (upper left of the window) presents an overview of the model hierarchies: the hierarchy of systems corresponds to the organizational hierarchy and the hierarchy of actions to the functional hierarchy. With the tree panel, the modeler directs the CAD tool to generate a particular diagram from the model. For example, in Figure 2, ProductMarket is marked as a composite (i.e. vertical symbol on the left side of ProductMarket) and Customer and SVN are wholes (i.e. horizontal symbol on the left side of there names). The resulting diagram is on the right of the editing window. If the modeler expands the symbol before the lifecycle interaction in the tree panel, she instructs the CAD tool to represent the next functional level: the lifecycle is represented as an activity, and not anymore as an action. Figure 3 shows the corresponding diagram. Similarly, if the modeler makes the SVN composite, a diagram similar to Figure 5 is obtained (but at a less detailed functional level). The property panel (lower left of the window) shows the characteristics of

the selected model element. These characteristics are useful, for example, to store traceability information.

The diagram visible in Figure 2 represents the working object `ProductMarket` seen as a composite. In SEAM models, *working objects* always represent systems. In `ProductMarket`, we have two component working objects: `Customer` and `SVN`. They are both represented as a whole. In between the `SVN` and the `Customer`, there is an action `lifecycle`. This represents the overall behavior, within `ProductMarket`, in which the two systems (i.e. `SVN` and the `Customer`) participate. In `SVN`, its state and behavior are represented. At the selected functional level, only one action is visible in `SVN`: it is the `Lifecycle` action. The state and behavior of the `Customer` are hidden. The importance of the lifecycle concept and the taxonomy of actions are further discussed in Section 3.

2.2. Terminology and Notation

Figure 3 shows the same systems as Figure 2. But the behavior of `SVN` is represented at a more detailed functional level. The `Lifecycle` activity (in Figure 3) replaces the `Lifecycle` action in Figure 2. We define as *functional refinement* the relation between an action and its corresponding activity. The action is always considered as a whole and the activity is always considered as a composite (i.e. composed of actions with execution constraints between the actions). The fact that an action corresponds to an activity is defined by the behavioral equivalence between the activity and the action: it is possible to replace the description of the action by the description of the activity without changing the overall system's behavior [7]. The level of atomicity of the behavior (i.e. which action is represented) defines the *functional level*. The descriptions of a system's behavior at different functional levels constitute the *functional hierarchy* of the system.

In the diagram (Figure 3), we represent `SVN`'s behavior and state (state related pictograms are rectangles; behavior related pictograms are shapes with rounded corners). We detail the behavior representation and then the state representation of `SVN`.

In the behavioral representation of `SVN`, the `SVN`'s `Lifecycle` is represented as an *activity*. It is composed of a `Begin` internal action (corresponding to the initialization action, executed first at system's creation), followed by a `Sell` partial interaction and an `End` internal action (corresponding to the termination action, executed last at system's disappearance). A *partial interaction* is an action that involves the working objects found in the environment of the working object that executes the partial interaction. An *internal action* is an action executed without involving the environment of the working object that executes the internal action. These partial interactions and internal actions, when represented in an activity, are separated by *execution constraints*. The execution constraint between `Begin` and `Sell` indicates that `Sell` can execute only when the *input environment parameters* `id`, `orderer` and `commit` or `cancel` are received. For simplicity, we model a system that can execute only one occurrence of `Sell`. So after `Sell`, `End` has to execute.

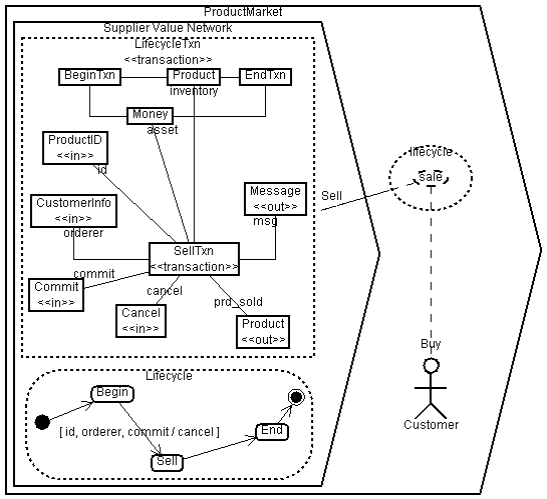

Figure 3: `SVN` does the `Lifecycle` activity (that includes the `Sell` action); `Customer` and `SVN` participate to the `sale` interaction.

In the state representation, we have *stateful* and *stateless properties*. Stateful properties represent the system state. Stateful properties can be *global* or *local*. Global properties exist for the whole system's lifecycle. For example, `assets` (of type `Money`) and `inventory` (of type `Product`) are global properties. Some properties are *local* and exist in the context of specific actions or activities. For example, `id` (of type `ProductID`) and `msg` (of type `Message`) are properties local to `SellTxn`. We will see in next section that local properties are also useful for storing information between actions within an activity (e.g. `Order` in `SellTxn` in Figure 4)

In SEAM, we have stateless properties. They are called *transactions* and they represent the occurrence of an action. They make explicit the context in which properties exist. For example: the `SellTxn` transaction represents the occurrence of the `Sell` action. Having transactions is useful for describing how properties relate to the actions. This is done through *transaction-property relations*. There are two kinds of transaction-property relations: environment parameters and system parameters. Each one can be further subdivided into input, output and in/out.

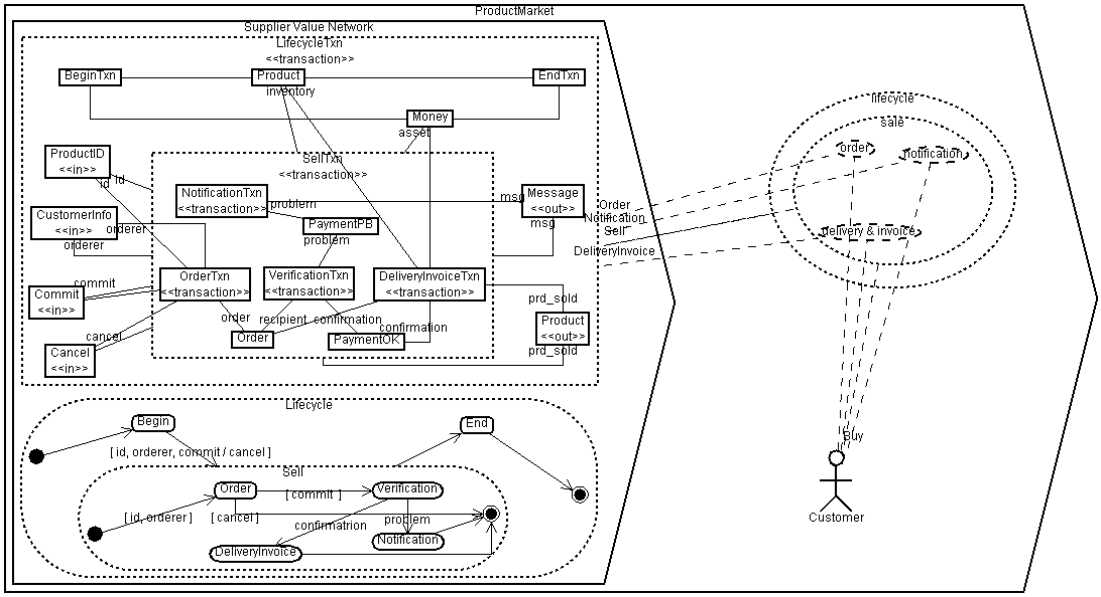

Figure 4: SVN does the Sell activity (that includes 3 partial interactions and the Verification internal action); Customer and SVN participate to 3 full interactions.

For example, SellTxn has an in/out system parameter relation to inventory (of type Product) and an input environment parameter relation to id (of type ProductID).

A transaction-property relation means that the pre- and post-condition of an action access the designated property. The property is then considered as a *parameter* for the action. *Environment parameters* are properties within a system that represent the state of the system's environment and that are necessary for exchanges through the system's boundary. *System parameters* are properties within a system that are not related to the system's boundary.

In Figure 3, we also provide behavioral information about ProductMarket: in ProductMarket, there is a full interaction between SVN and the Customer. A *full interaction* is an action in which multiple working objects participate. A behavioral equivalence exists between the full interaction and the composition of some (or all) of the partial interactions and of the internal actions of all working objects that participate in the full interaction. The role of SVN in Sale consists of the partial interaction Sell. The role of Customer in Sale consists of the partial interaction Buy. Both roles are visible in the *full interaction to working object relations*. These relations mark the working object's participation in the full interaction and the existence of a partial interaction. The benefit in representing Buy, Sale and Sell is to make the behavior of the different participants explicit. In ProductMarket, the Customer does a Buy, the SVN does a Sell and together they do a Sale. In the current version of SEAM in EA, it is not possible to express execution constraints between full interactions.

2.3. Functional Level Traceability

Figure 4 represents the same systems as Figure 2 and Figure 3. The behavior of SVN is represented at an even more detailed functional level. The Sell is now an activity that corresponds to the Sell partial interaction in Figure 3. The Sell activity is composed of the Order partial interaction, followed by the Verification internal action and by either the DeliveryInvoice partial interaction (if confirmation) or by the partial interaction Notification (if problem). The *conditional execution constraints* express conditional transitions. In this example, the transition between Verification and DeliveryInvoice is executed only when a confirmation exists.

The nesting of the notation makes visible in which context model elements exist. For example, Order exists in the context of Sell (or, by definition of the transactions, OrderTxn exists in the context of SellTxn). This nesting of notation is useful to understand how functionality is mapped between functional levels. For example, SellTxn (as an action) modifies inventory. Within the SellTxn (as an activity), it is DeliveryInvoiceTxn that actually modifies the inventory. Being able to compare the representations of the SellTxn as an action and as an

activity constitutes the *traceability between functional levels.*

2.4. Organizational Level Traceability

Figure 5 represents the SVN as a composite (as opposed to as a whole in Figures 2, 3 and 4). It is then possible to understand how the companies that compose the SVN interact to perform the behavior described for the SVN as a whole. We define as *organizational refinement* the relation between a working object as a whole and a working object as a composite (composed of component working objects considered as wholes). The level of atomicity of the working objects defines the *functional level*. We define the *organizational hierarchy* as the systems' hierarchy in the model (e.g. the working object ProductMarket made up of the working objects SVN and Customer, SVN working object is itself made up of the MarketingCo, ShippingCo and ManufacturingCo working objects).

In Figure 5, we have two behavior representations within the SVN working object: one with full interactions and one with partial interactions. We explain both of them below.

The representation with the full interactions makes explicit what the working objects do together. For example, the full interaction market involves only the participation of MarketingCo; the full interaction shipping_notice involves the participation of the three companies; shipping involves the participation of only ShippingCo, etc… The full interactions are useful for relating the behaviors described at different organizational levels. For example, the full interaction market in the SVN as a composite corresponds to the partial interaction Order in the SVN as a whole (Figure 4). The comment makes this relation explicit. This is the *traceability between organizational levels*. Full interactions are also useful to describe the net effects of a behavior without giving the details of the interaction between the systems (in the manner made popular by Catalysis [8] with the concept of joint actions).

The representation of partial interactions makes explicit the behavior of each working object relative to its environment. For example, MarketingCo gets id (of type PID) and info (of type OrdererInfo) in the partial interaction Market. As written in the comments, these two parameters are implementations of the ProductID and CustomerInfo represented in SVN as a whole (Figure 4). MarketingCo processes these parameters. When it receives commit, it executes Invoice. If a problem is detected during the execution of Invoice, then MarketingCo executes Notification. If the payment is ok, then MarketingCo modifies the value of the asset and requests the shipping by executing Shipping_notice. This sends an Order to ShippingCo and ManufacturingCo. Upon reception of the order, ShippingCo records this information in addr (of type ShippingInfo) and waits for the delivery (of type productDelivered). Upon reception of the order, ManufacturingCo takes prd (of type Product) from the inventory (of type Product) and sends delivery (of type ProductDelivered) to ShippingCo. ShippingCo, when getting the delivery, uses the shippingInfo to send the prd_shipped (of type ProductToBeShipped). As written in the comment, this parameter is an implementation of Product in the SVN as a whole (Figure 4).

The traceability between the SVN as a composite executing the Sell activity and the SVN as a whole executing the Sell activity (Figure 4) is visible with the comments marked *implementation of*. Three kinds of traceability relations exist. They establish either: (1) the relation between the partial interaction of a working object as a whole and the full interaction between its component working objects (e.g. Order in SVN in Figure 4 becomes market in Figure 5), or (2) the relation of a global property of a working object as a whole with the property of its component working object (e.g. inventory in SVN in Figure 4 becomes ManufacturingCo's inventory in Figure 5), or (3) the relation between a parameter in a working object as a whole and a parameter in a working object as a composite (e.g. product <<out>> in Figure 4 that becomes ShippingCo's ProductDelivered <<out>> in Figure 5). Traceability is not only between functional and organizational levels but also between systems at the same functional level and organizational level. For example, Order <<out>> in ManufacturerCo corresponds to Order <<in>> in ShippingCo and ManufacturingCo. Another example: ProductDelivered <<out>> is sent to the Customer. This kind of relation is not shown in the diagrams but can be captured in the CAD tool as a characteristic of the parameter (visible in the property panel of the editing window of the CAD tool).

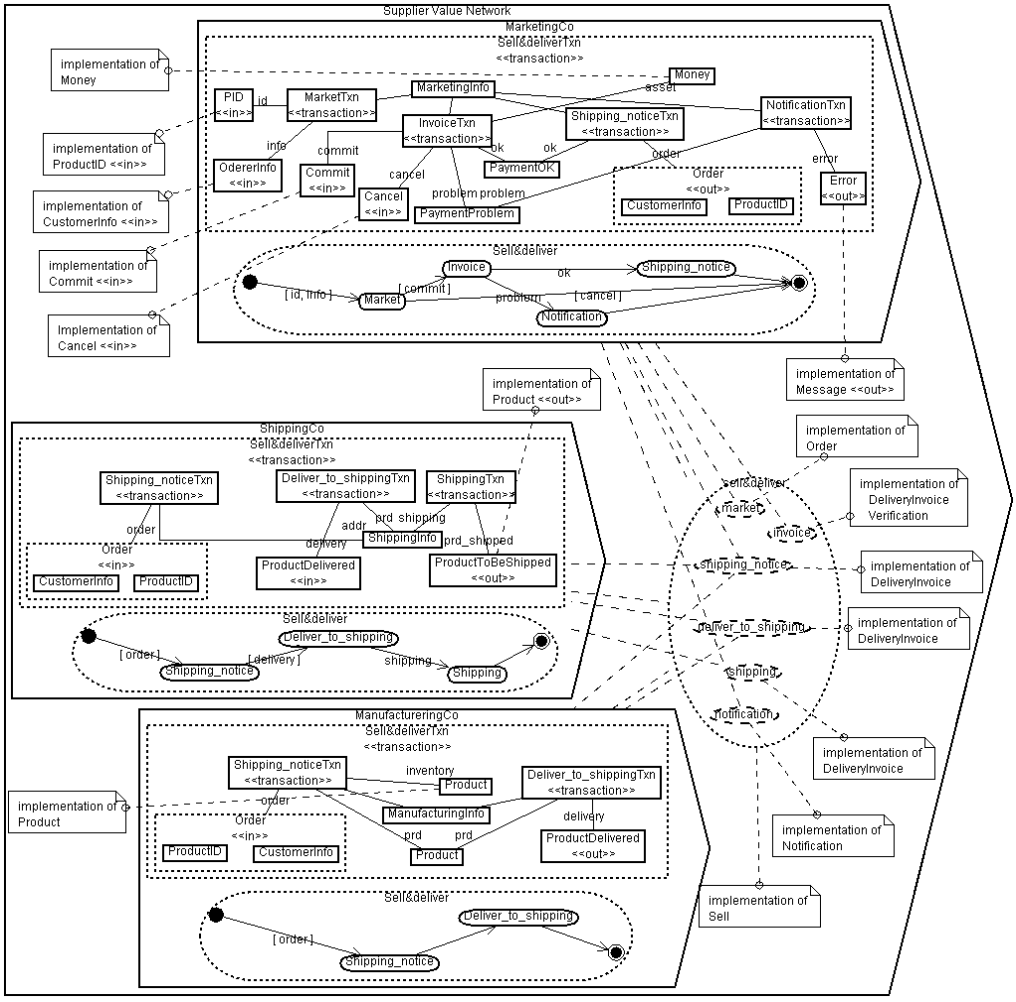

Figure 5: Diagram representing `MarketingCo`, `ManufacturingCo` and `ShippingCo` doing the `Sell` activity.

3. Applicability of RM-ODP Part 2 for Multi-Level Modeling

One of the challenges when building a modeling tool is the modeling ontology. An ontology defines the terms and the relations between these terms. These definitions are necessary to build a model. Examples of terms are: working object, behavior, action, activity, state, etc… Example of relations between terms: objects have state and behavior. In this section, we discuss the applicability of RM-ODP Part 2 (foundations) as such an ontology and we propose some extensions to support multi-level modeling.

The RM-ODP standard [5] is composed of four parts. Part 1 is an overview of RM-ODP and is non-normative. Part 2 defines the fundamental concepts needed for modeling of Open Distributed Processing systems. Part 3 presents an application of Part 2 for particular viewpoint specification languages (e.g. enterprise, information, computational, technology, engineering viewpoints). Part 4 is a partial formalization of the previous parts. In this work, we focus on Part 2.

Part 2 of RM-ODP [5, Part 2] has 15 sections. Sections 1 to 4 introduce the context, references and abbreviations. Section 5 introduces the categorization of ODP concepts. This section is needed to understand how all the following sections relate to each other. Sections 6 (*basic interpretation concepts*), 8 (*basic modeling concepts*) are central to this work and are discussed in this document. Section 9 (*specification concepts*) defines the terms necessary to specify the basic modeling concepts (e.g. type and instance). As these concepts are compatible with our approach, we do not discuss them in this paper. Sections 7 and 10 to 15 define supplementary concepts that are beyond the scope of this work.

3.1. Basic Interpretation Concepts

The Section 6 of [5, Part 2] *basic interpretation concepts* introduces the concepts needed for the

interpretation of the terms defined in the standard. When modeling, the modeler finds interesting *entities* in the *universe of discourse* and represents them as *model elements* in the *model*. Note that the term *model element* is not defined in RM-ODP. It is a concept, defined in [9], that can be considered as a specialization of the concept of *term* defined in [5, Part 2, Clause 7.1]. Figure 6 illustrates the relationships between the universe of discourse and the model and the relationships between the model and the diagrams. The CAD tool manages the model and the diagrams.

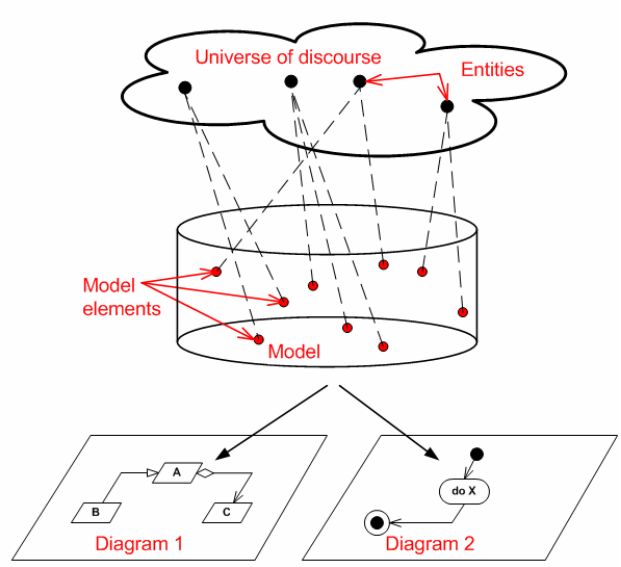

Figure 6: Relations between universe of discourse, model and diagrams [10].

Model elements are defined by a basic modeling concept and one or more specification concepts. In order to interpret the relations between these two kinds of concepts, we can apply Russell's Theory of Types in the way that is explained in [9], [11]: at first a model element is considered as something destitute of complexity. Then a first-order predicate is applied to this model element – this gives a possibility to specify the essence of the model element in the resulting first-order proposition. And then higher-order predicates are applied on top of the first-order proposition – this gives a possibility to construct higher-order propositions containing all the different characteristics of the specified model element. The aforementioned first-order predicate indicates the nature of the model element. The concepts that correspond to the first-order predicates can be found in the *basic modeling concepts* section of RM-ODP [5, Part 2, Section 8]. Examples of basic modeling concepts are *object* and *action.* Then, the higher-order predicates characterize the model elements within the higher-order propositions. The concepts that correspond to the higher-order predicates can be found in the *specification concepts section* of RM-ODP [5, Part 2, Section 9]. Examples of specification concepts are *type* and *instanc*e. To summarize with an example, an `action type Sell` can be understood as a model element with a first-order predicate `action` and a higher-order predicate `type Sell`.

To be able to understand the basic modeling concepts, modelers need to share (explicitly or implicitly) an agreed conceptualization of what represents the basic modeling concept in the universe of discourse. For example: What is an action? This agreed conceptualization is the Tarski declarative semantics. For example, in RM-ODP, *action* is explicitly defined as *something which happens* [5, Part 2, Clause 8.3]. *Something which happens* is the conceptualization of the universe of discourse, agreed by ODP modelers (as we assume that the modelers agree on what - *something which happens* - means). *Action* is the representation of this conceptualization in the model. RM-ODP defines another agreed conceptualization, which is *system*. *System* is defined as *something of interest as a whole or as comprised of parts* [5, Part 2, Clause 6.5]. However, RM-ODP does not define explicitly a basic modeling concept that has *system* as conceptualization. We discuss this point in the next section.

In [5, Part 2, Section 6] the terms *abstraction* and *atomicity* are explained. In particular, it is written that *fixing a given level of abstraction may involve identifying which elements are atomic*. In SEAM, we define two kinds of levels of abstraction: the functional levels and the organizational levels. The functional levels address the system's behavior. A functional level is defined by which action is considered as atomic (see Section 2.2 for an example). The organizational levels address the systems' construction. An organizational level is defined by which system is considered as atomic (see Section 2.4 for an example). These notions of levels take their roots in constructivism: constructivism states that all knowledge is relative to the observer [12] [13]. Observer-independent descriptions of reality do not exist. Different functional levels and organizational levels correspond to the different abstractions that the different kinds of observers have developed to simplify their understanding of systems. It happens that these abstractions appear hierarchical and this is why we call them functional and organizational hierarchies. The functional hierarchy is frequently made explicit in system design. The organizational hierarchy is more rarely made explicit, as people consider it obvious. This can lead to ambiguities. We took the concept of organizational hierarchy from Miller's Living System Theory [14]. Miller has shown that a living system can be modeled systematically and hierarchically (from organization, made of groups, made of humans, made of organs, made of cells). We use a similar approach for enterprises. We model segments, made up of value networks, made up of companies, made

up of departments, made up of people, etc… In Section 5, we discuss the practical benefits and drawbacks of modeling an enterprise with two distinct hierarchies.

3.2. Basic Modeling Concepts

[5, Part 2, Section 8] defines the basic modeling concepts such as: object, action, state, etc… We present the minor extensions that we have made to develop the SEAM in EA approach.

First we have defined the notion of *working object*. The term working is added only to remove ambiguities with the other usages of the word *object*. In RM-ODP, the agreed conceptualization associated to the object concept is *model of an entity* [5, Part 2, Clause 8.1]. In SEAM, the object concept is specialized and we consider that it is always a *model of a system*. In our example, `SVN`, `Customer`, `ProductMarket`, `MarketingCo`, `ManufacturingCo` and `ShippingCo` are all working objects (and so are all perceived as systems in the universe of discourse). As a system is a kind of entity, this extension is compatible with the standard as written.

Secondly, we have refined the definition of the different kinds of action. In RM-ODP, actions are divided into internal action and interaction. In [5, Part 2, Clause 8.3], it is written: *an internal action always takes place without the participation of the environment of the object.* The other actions are interactions. To model systems as we propose, we need two kinds of interactions: full interaction and partial interaction. A *partial interaction* is an action of one working object of interest (represented as a whole) and that involves one or more working objects from in its environment. A *full interaction* is an action of one working object of interest (represented as a composite) and that involves one or more of its component working objects and that may or may not involve working objects in the environment of the working object of interest. In Figure 7, actions `M_S`, `R_A` and `TinR_A` are full interactions; actions `M_A`, `M_B`, `R_C`, `R_D`, `R_E` and `TinR_C` are partial interactions and action `U_C` is an internal action. In addition, a partial interaction corresponds to the participation of a working object in a full interaction. For example, the partial interaction `M_A` is the participation of `AinS` in the full interaction `M_S`. A full interaction might (or might not) involve the environment of the system that hosts the full interaction. For example, in Figure 7, the full interaction `R_A` of `AinS` is actually exchanging information with the environment of the system `AinS` (as `R_A` implements `M_A` which is a partial interaction that exchanges information with `BinS`). On the other hand, `M_S` does not have exchanges with the environment of `S`. To differentiate between these two kinds of full interactions, we define the *full local interaction* (that does not exchange information – as M_S) and *full non-local interaction* (that does exchange information - as R_A does).

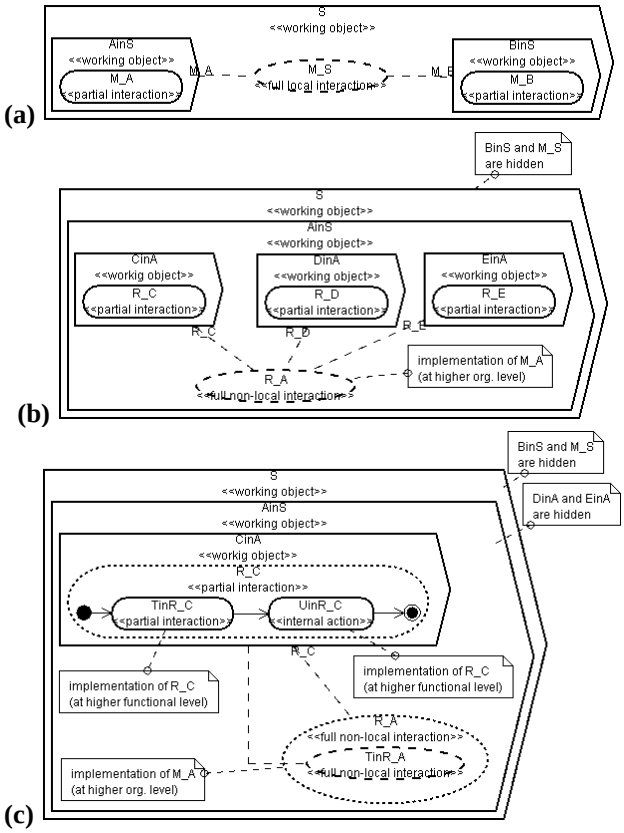

Figure 7. Examples of actions and traceability relations between organizational levels (a) and (b); between functional levels (b) and (c).

Are the concepts of full and partial interactions compatible with RM-ODP Part 2? Partial interaction is clearly defined in [5, Part 2, Clause 8.3]: *an interaction takes place with the participation of the environment of the object*. In addition the note 3 of [5, Part 2, Clause 8.3] states that *interactions may be labeled in terms of cause and effects relationships between the participating objects*. This hints that full interactions can also be considered compatible with RM-ODP Part 2 as interactions can happen between multiple objects. This is a point that the RM-ODP standard, in one of its future revisions, could make clearer.

Figure 8 summarizes these different kinds of actions. The taxonomy we propose makes explicit the context in which actions are defined (e.g. partial interactions and internal actions are defined for working objects as wholes) and the relations between the actions and the system boundaries (e.g. partial interactions and full non-

local interactions exchange information through the boundary of the working object of interest).

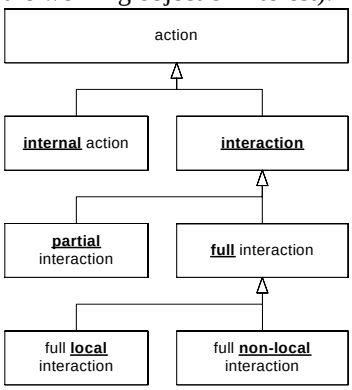

Figure 8: Proposed taxonomy of actions

Thirdly and last, we have introduced concepts necessary to structure the state space. In [5, Part 2, Clause 8.7], RM-ODP defines the concept of state as *at a given instant in time, the condition of an object that determines the set of all sequence of actions in which the object can take part*. The goal is to describe the state at the same level of details as the behavior. For this reason, it is important to add a means to structure the state. This is the concept of property. Properties can be stateless or stateful.

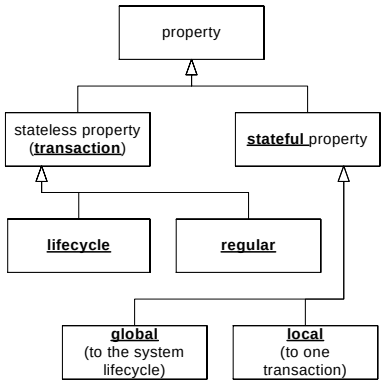

Figure 9: Proposed taxonomy of properties

Stateless properties represent occurrences of actions. Stateless properties are called transactions. They are similar to the stateless objects presented in [15]. One special transaction is the *lifecycle transaction* that represents the overall working object lifecycle. Transactions are useful to represent the context in which stateful properties exist.

Stateful properties store the system's state. They are similar to UML attributes except that they can be hierarchic (properties can be composite as well). Global properties exist in the context of the system lifecycle. They are created at system's initialization and disappear at system termination. Local properties exist in the context of a transaction.

Figure 9 summarizes the different kinds of properties. Beside stateless and stateful, the taxonomy we propose makes explicit in which context the property exists.

In summary, it appears that RM-ODP Part 2 concepts, as defined, can be applied in multi-level system modeling. It would be still helpful to make minor adjustments to the standard to make this possibility more explicit.

4. Related Work

The main part of the state of the art compares the SEAM in EA approach with the existing RM-ODP based approaches (Section 4.1). To be complete, we also mention non RM-ODP related, approaches for multi-level modeling. In particular, we present EA methods (Section 4.2) and software-engineering methods (Section 4.3).

4.1. ODP-Related Approaches

To our knowledge, the SEAM in EA approach, which uses directly the RM-ODP Part 2 concepts, is unique. Other approaches are based on RM-ODP viewpoints as defined in Part 3. For example, they define viewpoint languages (e.g. [16]), check consistency between viewpoints (e.g. [17], [18], [19], [15]), map viewpoints to UML (e.g. [20]) or develop EA CAD tools based on viewpoints (e.g. [21]).

It is worth comparing the SEAM in EA approach with what the viewpoint languages and the corresponding specifications provide [5, Part 3]. First of all, we can consider that we mix together information that is traditionally found in different viewpoints specifications. For example, Figure 3 can be interpreted as including the information found in the computational specification of `ProductMarket` mixed with the information found in the information specification of `SVN`. As we design multiple systems at the same time (`ProductMarket` in parallel with `SVN`) and as we always represent the contextual information (`ProductMarket` as context for `SVN`), it is not surprising that both kinds of information exist in the same diagram. Note that, by filtering the diagram, it is possible to hide part of this information and so, to become closer to the traditional RM-ODP viewpoints.

If we state that the definition of a system as a whole is close to an information specification, it is worth detailing how the concepts defined in the information viewpoint exists in SEAM in EA. The information viewpoint is defined in terms of schema (static, dynamic, and invariant). None of these schemas are presented in this paper. However, the SEAM relations between the transactions and the properties are useful to capture pre- and post-conditions of actions and thus, are related to a dynamic schema. The state part of the system

specification has information similar to an invariant schema. The future work (Section 5) will make these parallel more explicit.

SEAM in EA targets system modeling at large and, in particular, enterprise modeling. Thus, it is important to mention the enterprise language standardization [22] that refines and extends the enterprise language as defined in [5, Part 3]. This standard addresses enterprise modeling and the work presented in this paper is related to the concepts defined in it. For example, we could consider the notion of working objects as similar to the concept of community object. However, the enterprise language standard has concepts in deontic logic that SEAM does not provide.

The RM-ODP Part 3 approach to system specification is, in some degree embodied in the UML Profile for EDOC. This profile is composed of 7 standards [23] (overview, meta-model for Java and EJB, flow and collaboration specifications, pattern and relationship and relation to MOF). The UML Profile for EDOC and SEAM in EA share the same goal: to describe an enterprise. The main difference is in the ontology selected to express the models. In SEAM, our goal is to be as simple as possible, so that we stay as close as possible to RM-ODP Part 2. In the UML profile for EDOC, the goal is to be as close as possible to UML while making use of RM-ODP concepts. As UML is more complex than RM-ODP [24], the result is more complex. Strong parallels can be established between SEAM in EA and the UML profile for EDOC. For example, the definition of patterns can be compared to [25] and the definitions of relationships in UML for EDOC to the relations defined in SEAM in EA.

4.2. Multi-level Modeling in Business and EA

There are a significant number of methods applicable in EA that support some form of multi-level modeling. Our analysis shows that most methods do not provide a modeling ontology as described in this paper. Most of them (such as [26]) propose ad-hoc modeling frameworks. An exception is DEMO (Design & Engineering Methodology for Organizations) [27]. DEMO does provide such an ontology. The DEMO ontology is rooted in the Communicative Action Paradigm, regarding human communication and action. DEMO defines 3 types of models of the system: the black-box model, the white-box model, and the flow model. The black-box model deals mainly with the external behavior of a system and supports the functional decomposition mechanism. In the flow model a system is conceived as a network of nodes transforming the input flows into output flows. The white-box model defines the constructional decomposition of the system. It specifies the definition of subsystems [27]. SEAM in EA differs from DEMO in its goal (modeling more organizational level) and in the ontology used (RM-ODP instead of Communicative Action Paradigm).

4.3. Multi-level Modeling in System and Software

There are also numerous methods developed for multi-level modeling in system engineering and in software engineering. The closest to the SEAM in EA approach are:

OPM (Object-Process Methodology) [28] addresses the modeling of systems in general. It has a notation and a CAD tool called OpCat [29]. SEAM differs from OPM by its ontology (which is RM-ODP based) and by its explicit emphasis on the need to design multiple systems concurrently.

Catalysis [8] is a development process that analyzes and designs in three levels: business, IT system and software components. It uses its own UML-inspired notation. SEAM was inspired by Catalysis. The goal for SEAM in EA is to provide a design method analogous to Catalysis, but with a broader scope (from business down to IT) and based on RM-ODP.

SysML [30], developed by OMG, is a refinement of UML that targets the design of systems in general (e.g. aircraft) using the UML notation. KobrA [31] proposes a recursive model that describes IT systems/components using the UML notation. Both KobrA and SysML differ from SEAM in EA by their tight link to the UML meta-model (as opposed to RM-ODP). Even if both methods can model multiple systems, they are designed to focus mainly on one system of interest.

Domain-specific languages (such as Microsoft DSL [32]) automate software development as much as possible by defining expert engines that can generate code for a specification in very specific domains (e.g. banking, cell phone…). Tools are often based on GME [33] or Eclipse EMF [34]. Domain specific methods do not manage the transition between the different organizational levels in a similar way as SEAM in EA does. They relate directly the code to the domain mode. This is something SEAM in EA cannot do without modeling all organizational levels between the market organizational level and the IT system organizational level.

5. Applicability and Further Work

The method presented in this paper focuses on functional analysis of companies' environment, of companies' organization and of IT systems. This is a reductionist view of an enterprise. Analyzing

functionality across organizational levels is only a subset of what needs to be analyzed when designing an enterprise. For example, different specialists might focus on non-functional properties (such as performance or security). However, our experience shows that modeling function adds value as it defines a common, minimal, understanding across the whole organization.

Here are examples of projects using SEAM in EA:

- IT System Reengineering: A mid-size organization has to streamline its IT organization across product lines. This is a 5 years project that involves the whole company and multiple consultants. SEAM in EA is successfully used to represent the roles of the company in its market, the roles of the company's departments and the way the business processes need to be structured. The benefits are the development of a standardized terminology and of a visual model that can be used by the CIO in his decision making process. This project is described in [35].
- Project Documentation: A software company won a contract for a relatively large development of an IT application to manage taxes (approx. 4 years, 20 developers). SEAM in EA is successfully used to represent the project team structure and the application structure (in complement to the RUP design process). The goal for the SEAM model was to speed-up the training of new software developers. An on-line documentation system was developed and SEAM in EA diagrams are used to access the documentation. Experience has shown (on 3 people) that the training of the developers is reduced from 6 weeks to 2 weeks.
- Project Specification: SEAM in EA was used in a regular architectural project that lasted 18 months. The goal was to equip a building for a university. A SEAM model was developed to specify the goal for the equipment. This model was used to develop the business case and to specify to the vendors what needed to be provided (furniture, multi-media systems, IT systems…). It was also possible to generate a complete IT specification aligned to the business specification.

In all these projects, the SEAM model was useful for agreeing on what systems exist and on the functionality provided by each one. Once this was agreed upon, the different specialists had fewer difficulties in communicating with each other and used their common understanding in developing their own models. This explains why the hierarchical nature of the SEAM model is not an issue as it is only considered as a shared model that all specialists can refer to in developing their own models.

Our future research work has two main directions: further evaluation of the approach with additional projects and more formal definition of the semantics of our notation. For this, we have three projects: (1) formal definition of static, dynamic and invariant schemas in SEAM [2] – similar to the schemas defined in [5, Part 3, Clause 6.1]. Our schemas have a declarative semantics based on Alloy [36]. (2) behavioral simulation and alignment checking [7] (with an operational semantics based on ASML [37]); (3) synthesis of the results of (1) and (2) in a formal model, in Alloy, of the SEAM in EA ontology. This Alloy formal model will be automatically translated into the Java code used in the SEAM CAD tool. This does guarantee that the tool implements rigorously what is defined in the SEAM ontology.

6. Conclusion

We have shown in this paper that the concepts defined in RM-ODP Part 2 (foundations) are well suited to multi-level system modeling. We also made suggestions on how the RM-ODP Part 2 definitions (and associated notes) could be modified to make more explicit that RM-ODP Part 2 can also be used directly to model systems without using the viewpoints defined in Part 3. Concretely, we have defined two new kinds of levels of abstraction (organizational and functional), we have recommended relating objects to systems and we have proposed new kinds of actions and properties. It could be useful, if a revision of the RM-ODP Part 2 standard is realized, to consider adding concepts such as properties or to define the concept of action more broadly to clearly encompass full and partial interactions.

We have also shown that concrete methods and tools can be developed, based on RM-ODP Part 2 directly. We have illustrated, with an example in Enterprise Architecture, how such methods could work. Our experience has shown that RM-ODP Part 2 can be used systematically on all modeling levels and that it defines concisely, and with precision, what the ontology for object-oriented modeling can be. Methods based on RM-ODP (such as SEAM) benefit from a powerful ontology definition that is, in addition, standardized.

We believe that the work we do with SEAM can also contribute to the RM-ODP community. First, our tool can be used to explain the concepts found in RM-ODP Part 2. Many of these concepts are obvious for people with experience in formal methods. However, our tool can illustrate these concepts graphically for the people who do not have such training. For example, our tool can be used to explain the difference between an internal action and an interaction or between the different kinds of refinement. Secondly, we propose a new way to model systems with RM-ODP and our case studies illustrate concretely how such RM-ODP based specifications of systems might look. This can contribute to the promotion

of RM-ODP. Lastly, we also consider that the CAD tool [3] we are developing is one of the rare tools that directly use the RM-ODP Part 2 concepts. This also contributes to make RM-ODP more visible.

References

[1] Luftman, J. and McLean, E. R., "Key Issues for IT Executives," *MIS Quarterly Executive*, vol. 3, 2004.
[2] Wegmann, A., "On the Systemic Enterprise Architecture Methodology (SEAM)," presented at 5th ICEIS, Angers, France, April 2003.
[3] Lê, L. S. and Wegmann, A., "SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture," presented at 39th IEEE HICSS, Hawaii, USA, January 2006.
[4] OMG, Unified Modeling Language, http://www.uml.org/
[5] OMG, "ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation, X.901, X.902, X.903, X.904, Reference Model of Open Distributed Processing," 1995-1996.
[6] Lê, L. S. and Wegmann, A., "Definition of an Object-Oriented Modeling Language for Enterprise Architecture," presented at 38th Hawaii International Conference on System Sciences, Hawaii, USA, January 2005.
[7] Wegmann, A., Balabko, P., Lê, L. S., Regev, G., and Rychkova, I., "A Method and Tool for Business-IT Alignment in Enterprise Architecture," presented at 17th CAiSE Forum, Porto, Portugal, June 2005.
[8] D'souza, D. F. and Wills, A. C., *Object, Components and Frameworks with UML, The Catalysis Approach*: Addison-Wesley, 1999.
[9] Naumenko, A., *Triune Continuum Paradigm: a paradigm for General System Modeling and its applications for UML and RM-ODP*, PhD thesis, in School of Computer and Communication Sciences, EPFL, 2002
[10] Wegmann, A. and Naumenko, A., "Conceptual Modeling of Complex Systems using an RM-ODP based Ontology," presented at 5th IEEE EDOC, Seattle, USA, September 2001.
[11] Naumenko, A. and Wegmann, A., "Formalization of the RM-ODP foundations based on the Triune Continuum Paradigm," *Computer Standards & Interfaces - ELSEVIER*, 2006.
[12] Moigne, J. L. L., *Que sais-je? Les épistemologies constructivistes*. Paris: Presses Universitaires de France, 1995.
[13] Checkland, P. and Scholes, J., *Soft System Methodology in action*: Chichester UK: Wiley, 1990.
[14] Miller, J. G., *Living Systems*: University of Colorado Press, 1995.
[15] Bernardeschi, C., Dustzadeh, J., Fantechi, A., Najm, J., Nimour, A., and Olsen, F., "Transformation and Consistent Semantics for ODP Viewpoints," presented at FMOODS'97, Canterbery, UK, July 1997.
[16] Lupu, E., Sloman, M., Dulay, N., and Damianou, N., "Ponder: realising enterprise viewpoint concepts," presented at 4th IEEE EDOC, Makuhari, Japan, September 2000.
[17] Dijkman, R., Quartel, D., Pires, L., and Sinderen, M., "A Rigorous Approach to Relate Enterprise and Computational Viewpoints," presented at 8th IEEE EDOC, California, USA, September 2004.
[18] Boiten, E., Bowman, H., Derrick, J., Linington, P., and Steen, M., "Viewpoint consistency in ODP," *ELSEVIER*, 2000.
[19] Dustzadeh, J. and Najm, J., "Consistent Semantics for ODP Information and Computational Models," presented at FORTE / PSTV'97, Osaka, Japan, November 1997.
[20] Romero, R. and Vallecillo, A., "Modelling the ODP Computational Viewpoint with UML 2.0," presented at 9th IEEE EDOC, Enschede, The Netherlands, September 2005.
[21] Steen, M. W. A., Akehurst, D. H., Doest, H. W. L., and Lankhorst, M. M., "Supporting Viewpoint-Oriented Enterprise Architecture," presented at 8th IEEE EDOC, California, USA, September 2004.
[22] Miller, J., RM-ODP Enterprise Language, http://www.joaquin.net/cuml/Ent/index.html
[23] OMG, UML Profile for enterprise distributed Object Computing, http://www.omg.org/technology/documents/formal/edoc.htm
[24] Naumenko, A. and Wegmann, A., "A Metamodel for the Unified Modeling Language," presented at <<UML>> 2002, Dresden, Germany, September/October 2002.
[25] Balabko, P., Wegmann, A., Ruppen, A., and Clément, N., "Capturing Design Rationale with Functional Decomposition of Roles in Business Processes Modeling," 2005.
[26] Zachman, J. A., "A Framework for Information System Architecture," *IBM System Journal*, 1988.
[27] Dietz, J., Design & Engineering Methodology for Organizations, http://www.demo.nl
[28] Dori, D., *Object-Process Methodology, A Holistic Systems Paradigm*: Springer Verlag, 2002.
[29] Dori, D., Reinhartz-Beger, I., and Sturm, A., "OPCAT - A Bimodal CASE Tool for Object-Process Based System Development," presented at 5th ICEIS, Angers, France, April 2003.
[30] OMG, System Modeling Language, http://www.sysml.org/
[31] Atkinson, C., Paech, B., Reinhold, J., and Sander, T., "Developing and applying component-based model-driven architectures in KobrA," presented at 5th IEEE EDOC, Seattle, USA, September 2001.
[32] Microsoft, Domain-Specific Language Tools, http://msdn.microsoft.com/vstudio/DSLTools/
[33] Karsai, G., Maroti, M., Ledeczi, A., Gray, J., and Sztipanovits, J., "Composition and cloning in modeling and meta-modeling," *IEEE Transactions on Control Systems Technology*, vol. 12, pp. 263-278.
[34] Eclipse Modeling Framework, http://www.eclipse.org/emf
[35] Wegmann, A., Regev, G., and Loison, B., "Business and IT Alignment with SEAM," presented at REBNITA / 13th IEEE RE workshop, Paris, September 2005.
[36] Jackson, D., The Alloy Constraint Analyzer, http://alloy.mit.edu/
[37] Börger, E. and Stärk, R., *Abstract State Machines: A Method for High-Level System Design and Analysis*: Springer-Verlag, 2003.