SITUATION-BASED MODELING FRAMEWORK FOR ENTERPRISE ARCHITECTURE

THÈSE N^{O} 3234 (2005)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Institut d'informatique fondamentale

SECTION DES SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Pavel BALABKO

M.Sc. in Mathematical Engineering, St. Petersburg Technical University, Russie
et de nationalité russe

acceptée sur proposition du jury:

Prof. A. Wegmann, directeur de thèse
Prof. C. Atkinson, rapporteur
Prof. F. Bouchet, rapporteur
Prof. Ch. Tucci, rapporteur

Lausanne, EPFL
2005

Abstract

This thesis presents the Situation-Based Modeling Framework for Enterprise Architecture. This framework improves system modeling by making models more systemic and, therefore, making reasoning about these models easier.

The context of this thesis is Enterprise Architecture (EA). EA enables enterprises to anticipate or react to necessary business or technical changes. EA models have to represent enterprises facets from marketing to IT. As a result, EA models tend to become large and complex. To cope with the complexities of large enterprise models, they are usually considered as compositions of smaller, manageable parts: concerns. We call Concern-Based Design Methods (CBDMs) the methods that support modeling with concerns.

In order to understand how CBDMs can be used in EA, we have made the systemic analysis of twenty of the most influential CBDMs that can be used in the context of EA. In our analysis we have studied the ontological, epistemological, theoretical and methodological principles of the considered CBDMs. The result of the analysis shows that ontological and theoretical principles are often supported by CBDMs. These principles make CBDMs convenient for the management of systems' concerns: CBDMs explain how to define, store, compose and reuse systems' concerns. However, the support of methodological and epistemological principles in the existing CBDMs is often missing. In our work we concentrate on the support of the epistemological principles. Epistemology reflects the way human cognize. Without epistemological principles, CBDMs result in models that are difficult to understand and reasoning about.

In this thesis we present a Situation-Based Modeling Framework. The novelty of our framework is its foundation in a set of epistemological principles. These principles originate from our state of the art analysis of the existing CBDMs. They allow for building models that are more systemic and, therefore, more comprehensible. Between them the following principles we consider as the most important in the context of EA:

- Situation Relativity: a modeling method should support explicit modeling of situations.
- State and Behavior Holism: the state (or data) and behavior models should be integrated in the context of EA. This allows for identifying breakdowns in behavior/data interactions and for specifying explicitly the life cycle of information elements: i.e. when and where information elements are created, used and deleted.
- Diagrammatic representation: this principle is especially important in the context of EA. EA design models should be used by specialists with different backgrounds. These specialists have to communicate in languages that can be rapidly understood by all of them. Visual languages help to solve this problem.

The first principle plays the central role in the definition of our framework. In our Situation-Based Modeling Framework we propose to identify system concerns by means of considering a system of interest in a number of specific situations. Concerns are explicitly related to these situations. The whole behavior of a system is defined as the composition of these situation-related concerns. In order to make situations the first-class citizens of our modeling framework, we took the key principles for our framework from the situation theory. The situation theory is a mathematical theory of meaning which clarifies and resolves some problems in the study of language, information, logic, philosophy, and the mind. The direct application of this theory is not feasible in the context of EA because this theory is quite formal and too much adapted to linguistic needs. However, we took the basic principles from this theory as foundations of our situation-based modeling framework:

- Situation-based modeling: Any entity (a system) should be considered in a number of situations. Any assertion about system properties depends on a situation.
- Explicit modeling of constraints between situations: the mutual relations between concerns of the same system should be modeled explicitly. In our work we model these mutual relations with *composition constraints*. Using composition constraints allows a designer to make her decisions about the composition of concerns explicit.
- Hierarchical modeling of situations: The behavior of an enterprise system should be represented in the form of the role hierarchy that corresponds to the hierarchy of situations.

The main impact of this work consists in making systems' functional models more systemic. These models are more understandable, human friendly and comfortable for reasoning. We have confirmed this impact of our

modeling framework with three applications: in the domain of software engineering we used our framework as the basis for the Rapid Prototyping Tool based on Aspect-Oriented and Subject-Oriented programming. In the domain of business process modeling we applied our framework as the theoretical background for the conceptual tool for business process innovations and for the specification of a manufacturing process. This application for the specification of a manufacturing process was used as a practical validation of our framework. We used our framework for a project in a big pharmaceutical company. This company needed to introduce a MES (Manufacturing Execution System) to ensure the order of the manufacturing process. The project members acknowledged the ability of our framework to improve the comprehensibility of a manufacturing process.

Another important result of this thesis is a step forward in categorizing numerous CBDMs that can be used in the context of EA. We have defined the requirements for CBDMs. These requirements can be used: to evaluate, compare and choose appropriate CBDMs for using them in EA projects; to develop new CBDMs in the context of EA or to extend existing methods with concern-based modeling.

Version Abrégée

Cette thèse présente le cadre de modélisation appelé « Situation-Based » pour l'Architecture d'Entreprise. Ce cadre améliore la modélisation de systèmes en faisant des modèles plus systémiques et donc facilitant le raisonnement a propos de ces modèles.

Le contexte de cette thèse est l'Architecture d'Entreprise (EA). EA permet aux entreprises de prévoir ou réagir aux changements de l'environnement métier ou technique. Les modèles d'EA doivent représenter des facettes de l'entreprise du marketing à l'IT. Par conséquent, les modèles d'EA ont tendance à devenir larges et complexes. Pour faire face aux complexités de larges modèles d'entreprise, ces modèles sont d'habitude considérés comme des compositions de parties plus petites et maniables appelées « concerns ». Nous appelons « Concern-Based Design Methods » (CBDMs) les méthodes de conception qui permettent la modélisation basée sur les concerns.

Afin de comprendre comment les CBDMs peuvent être utilisé dans l'EA, nous avons fait l'analyse systémique de vingt CBDMs les plus influents qui peut être utilisé dans le contexte de l'EA. Dans notre analyse nous avons étudié les principes ontologiques, épistémologiques, théoriques et méthodologiques de ces CBDMs. Le résultat de l'analyse montre que les principes ontologiques et théoriques sont souvent utilisés par ces CBDMs. Ces principes rendent ces CBDMs commodes pour la gestion de concerns: Ces CBDMs expliquent comment définir, sauver, composer et réutiliser des concerns. Cependant, l'utilisation de principes méthodologique et épistémologique dans ces CBDMs manque souvent. Dans notre travail nous concentrons sur l'utilisation de principes épistémologiques. L'épistémologie reflète la façon dont l'humain construit sa connaissance. Sans l'épistémologie, l'utilisation de ces CBDMs résulte dans des modèles difficiles à comprendre et raisonner avec.

Dans notre thèse nous présentons le cadre de modélisation appelé « Situation-Based » pour l'Architecture d'Entreprise. La nouveauté de notre cadre de modélisation est son fondement dans un ensemble de principes épistémologiques. Ces principes ont pour origine notre analyse des CBDMs évoqués ci-dessus. Ils permettent de construire des modèles plus systémiques et donc plus compréhensibles. Nous considérons les trois principes suivants comme les plus importants dans le contexte de notre travail:

- Relativité par rapport aux situations: une méthode de modélisation doit utiliser la modélisation explicite de situations.
- Holisme de comportement et d'état: Les modèles d'état (ou de données) et de comportement doivent être intégrés dans le contexte d'EA. Ceci permet d'identifier des défaillances dans les interactions entre comportement et données et de spécifier explicitement le cycle de vie d'éléments d'information: c.-à-d. quand et où les éléments d'information sont créés, utilisés et enlevés.
- La représentation diagrammatique: ce principe est surtout important dans le contexte d'EA. Les modèles d'EA doivent être utilisés par des spécialistes dans différents domaines. Ces spécialistes doivent communiquer entre eux en utilisant des langages qui sont rapidement compréhensibles. Les langages visuels aident à résoudre ce problème.

Le premier principe joue le rôle central dans la définition de notre cadre. Dans notre cadre nous proposons d'identifier les concerns en considérant le système dans plusieurs situations spécifiques. Les concerns sont explicitement reliés à ces situations. Le comportement du système est défini comme la composions de ces concerns. Afin de donner une grande importance au concept de situation dans notre cadre de modélisation, nous avons défini les principes clés de notre cadre en prenant comme référence la théorie des situations. La théorie des situations est une théorie mathématique du sens qui clarifie et résout quelques problèmes dans l'étude du langage, de l'information, de la logique, de la philosophie, et de l'esprit. L'application directe de cette théorie n'est pas faisable dans le contexte d'EA du fait que cette théorie est particulièrement formelle et trop adaptée aux besoins de la linguistique. Cependant, nous avons pris les principes fondamentaux de cette théorie comme fondement de notre cadre de modélisation. Ces principes sont:

- La modélisation basée sur les situations: Toutes les entités (systèmes) doivent être considérées dans plusieurs situations. Toute affirmation concernant les propriétés d'un système dépend d'une situation.
- La modélisation explicite de contraintes: Les relations mutuelles entre les concerns d'un même système doivent être explicitement modélisés. Dans notre travail nous modélisons ces relations

mutuelles avec des contraintes de composition. L'utilisation des contraintes de composition permet à un concepteur de rendre ses décisions de la composition de concerns explicites.

- La modélisation hiérarchique: Le comportement d'un système d'entreprise doit être représenté sous la forme d'une hiérarchie de rôles qui correspond à la hiérarchie des situations.

L'impact principal de ce travail consiste à rendre les modèles fonctionnels des systèmes plus systémiques. Ces modèles sont plus compréhensibles, ergonomiques et confortables. Nous avons confirmé cet impact avec trois applications : dans le domaine d'ingénierie du logiciel nous avons utilisé notre cadre comme la base pour l'Outil de prototypage rapide basé sur la programmation orienté aspect et orienté sujet. Dans le domaine de la modélisation des processus métier nous avons utilisé notre cadre de modélisation en tant que fondement théorique pour l'outil d'innovation de processus métier et pour la spécification d'un procédé industriel. Cette spécification d'un procédé industriel a servi de validation pratique de notre cadre. Nous avons utilisé notre cadre pour un projet dans une multinationale de la pharmaceutique. Cette multinationale voulait introduire un système de suivi de fabrication assurant le bon fonctionnement du procédé industriel. Les membres de ce projet nous ont donné des commentaires positifs concernant la capacité de notre cadre à améliorer l'intelligibilité d'un procédé industriel.

Un autre résultat important de cette thèse est un avancement de la classification des nombreux CBDMs pouvant être utilisés dans le contexte d'EA. Nous avons défini les exigences pour les CBDMs. Ces exigences peuvent être utilisées pour : évaluer, comparer et choisir des CBDMs appropriés pour l'utilisation dans un projet d'EA ; pour développer des nouveaux CBDMs dans le contexte d'EA ou pour étendre les méthodes existantes avec la modélisation basée sur les concerns.

TABLE OF CONTENTS

PREFACE

The way we think and behave always depends on situations in which we are involved. For example, the human behavior is different in unfriendly and friendly environment situations. Humans change behavior depending on the community: the same behavior can be understood quite differently by humans in different communities. The understanding of situations is necessary for the correct interpretation of human behavior and ideas. Any human statement should be understood only in a relation to a certain situation. This is probably the main idea of this thesis that distinguishes our modeling approach from many other approaches.

From one side, situation-based modeling, i.e. modeling system behaviors explicitly with the situations where these behaviors are defined, is definitely a very new topic in the computer science disciplines. Traditionally in these disciplines, statements are interpreted in only in one way and models are considered as situation invariant. Therefore, saying that any statement should be interpreted in a relation to a certain situation brings something new in the field of system modeling.

From the other side we are not alone who try to model system behavior on the per-situation basis. During writing this thesis I have discovered a very interesting modeling framework called, Metapattern. The foundations and many ideas in this framework are very similar to what I have discovered during my work on this thesis. "As a matter of principle, there is no absolute, only contextual [situational] meaning. Different meanings may occur for the same object, presuming equally different contexts [situations]" [Wisse00]. These ideas definitely confirmed me that the direction this thesis is right. In the case with Metapattern, it was also interesting to see how from very similar with Metapattern basic ideas we got qiute a different modeling framework. This difference may be explained by the difference of the application fields. Metapattern is about information or conceptual modeling. Our framework is about enterprise architecture. We wanted to have models that allow a modeler to have a seamless bridge to business process modeling and software engineering.

Many ideas of thesis come from long discussions with my colleagues. This thesis conveys the spirit of members of our laboratory who tend to see any subject in many different situations or from different viewpoints. You can often hear during group meetings: "It depends how you see…", "From the other side…", "There is nothing unique". The discussions in our laboratory served me as the source of inspiration for this work. A lot of time was spent in trials to understand how to model systems in multi-situational environment. The origin of these discussions was an attempt to understand the concept of role: how to define and use roles in system modeling. The original name of this thesis was the Role-based Modeling framework. However with time, by giving different presentations and getting an output from them, I have realized that:

First of all, the concept of role is an "overloaded" term: there are too many approaches that assign quite different meanings to the role term;

Second, I realized that what we do more than simply "role modeling". In our laboratory we always try to explain the origin of all modeling elements. In the context of this work I have tried to understand the origin of roles models, i.e. where roles in the existing models are coming from. In our attempts to see this, we modeled roles by explicitly relating them to certain "situations of interest". These situations are typically related to system stakeholders: from system users to system engineers.

These two observations motivated me to change the name of this work to the Situation-Based Modeling Framework in the same time putting forward the concept of situation.

The important aspect that I want to mention before continuing with this thesis is that we did not aim to build a formal framework. The citation in the paper of M. Odeh, I. Beeson, S. Green J.Sa nicely expresses this idea: "The task is not so much to capture a process in order to automate it, as to comprehend a process ..." [Odeh02]. The important thing about this thesis is that it gives a flavor of thinking in terms of different situations and modeling system concerns related to these situations. I do not expect that our framework (especially its visual modeling language) can be fully used in the industrial projects. In many cases the visual models that we propose in this thesis would be too detailed. Therefore, for large industrial projects, our modeling language should be simplified or adapted to the existing visual languages (such as UML). The main value of our framework for large projects is to inspire the "situation-based reasoning" about systems and to help stakeholders to comprehend their systems better. In the three applications considered in this thesis we give examples that show how our modeling framework was used to inspire the situation-based reasoning about models.

It is also necessary to mention the Systemic Enterprise Architecture Methodology (SEAM) that was developed in our laboratory. Our framework is tightly correlated with SEAM: there are many common ideas between the Situation-Based Modeling Framework and SEAM; the language that we use in this thesis was heavily influenced by SEAM. However, in its current version SEAM does not fully support the situation-based modeling as proposed in this thesis. I believe that this thesis can serve as the basis for the extension of SEAM that will include the situation-based modeling. The full integration of our framework with SEAM and,

especially, with the corresponding SeamCAD tool is a subject of separate work and may result in a separate PhD work in the future.

In the end of this preface part I would like to acknowledge the efforts of all those who contributed to writing this work. First of all, I thank my PhD thesis director, Professor Alain Wegmann, the head of the Laboratory of Systemic Modeling (LAMS), who motivated me a lot during this work with a positive criticism and very useful comments. Especially, I am thankful to Alain Wegmann for a very nice atmosphere that he has managed to create in LAMS. It was really my pleasure to work to write the thesis in LAMS. Mainly because of the nice atmosphere it was possible to share ideas between members of our laboratory and get a permanent feedback on my on-going research work. I have appreciated a lot the help of my colleagues from LAMS mainly in the form of advises and their friendly support. In particular I am very thankful to Andrey Naumenko, Guy Genilloud, Gil Regev, Txomin Nieva, Otto Preiss, Irina Rychkova, Lam-Son Le and José Diego de la Cruz. There are also many other people, my friends and colleagues, working mainly in EPFL, who also contributed to writing this thesis. I want to thank all of them for their valuable help. I would also like to thank the jury members (Prof. Colin Atkinson, Prof. Christopher Tucci and Frederic Bouchet) who refined significantly the ideas of this thesis. And finally, I would also like to express my gratitude to my parents and my wife who have provided me a personal support during this work.

INTRODUCTION

CONTEXT

The context of this thesis is Enterprise Architecture (EA). “The primary reason for developing EA is to support business by providing the fundamental technology and process structure for an IT strategy” [OpenGroup03]. EA is a multi-disciplinary approach that enables enterprises to anticipate or react to necessary business or technical changes. In an EA project, the EA team develops *an EA model* (also called enterprise model) that represents the enterprise. The model is usually structured in hierarchical *organization levels*. The highest level typically describes marketing concerns, the middle level describes business processes, and the lower level describes the IT systems. The rationale behind structuring EA models with hierarchical levels can be found in [Wegmann03].

Usually, EA models become very large because they cover a very wide range of concerns from marketing down to IT implementation issues. As a result, EA models are often incomplete, inconsistent or unspecified. Enterprise models exemplify traditional modeling problems such as the one defined by [Clarke99]: “models are often large and monolithic”, “designs are too difficult to reuse” and “there is a significant structural misalignment between requirements and code, with design caught in middle”. This results in models that are difficult to understand, manage and, therefore, these models cannot be used for reasoning about or for designing business and IT systems. To cope with the complexities of enterprise models, they are usually considered as compositions of smaller, manageable parts: concerns. In order to do this, Concern-Based Design Methods (CBDMs) should be used. When we mention CBDMs, we refer to all design methods that represent design models as sets of concerns where each concern is an abstraction that reflects a part of a system relevant to a particular concept, goal, or purpose.

To understand how concerns can be used in the context of EA, we begin with an example (see Figure 1). This example is based on an enterprise model developed using the Systemic Enterprise Architecture Methodology (SEAM) [Wegmann03a]. It illustrates how an on-line company (*BookCo*) runs an EA project in order to gain profitability. The EA project is carried out by an EA team that includes specialists from different fields (from business to IT) and by one architect that federates the work. This team develops an enterprise model that has several organization levels representing the company. Each organization levels represent the views of a different specialist in the EA team. Let’s describe these levels as they are illustrated in Figure 1:

The Company organization level (Figure 1.a and 1.b) represents the companies working together to serve the end users. It includes *BookCo*, the company that that sells books, a publisher and a shipment company. We represent objects corresponding to these companies by nodes (boxes). Each organizational level may have several levels of detail. For example, the Company organization level in Figure 1 has two levels of detail. At the first level, the fact that the four companies are working together is represented by the Manufacturing and Sale (*Mfg&Sale*) collaboration (dashed oval in Figure 1.a). Each company plays a certain role in this collaboration. Roles are represented as stick men. At the second level, the *Mfg&Sale* collaboration is refined into three collaborations (*Purchase Selection*, *Purchase Ordering* and *Purchase Delivering*). Roles from level 1 are compositions of roles from level 2. For example, the Seller role of the *BookCo* (from level 1) object is a composition of the *Order Receiver* and *Select Service* roles (from level 2).

The IT Application organization level represents a set of IT applications that collaborate to fulfill roles from the Company organizational level. This level allows the IT specialists to talk about IT resources and their interfaces. For example, *BookCo* has three IT resources: the www server, the ERP system and a B2B application. These three resources fulfill the “Select Service” and “Order Receiver” roles from the Company organization level. The IT Application organization level is typically used to specify requirements for IT systems in a company. Note, that roles from the Company Organization level are compositions of roles from the IT Application Organizational level.

Organizational levels that go below the IT Application Organizational level represent a set of specific technological components that implement IT systems. For example these levels can be Data Base or Object Oriented Classes Organizational Levels. For the purpose of simplicity we do not show details about these levels in our example from Figure 1.

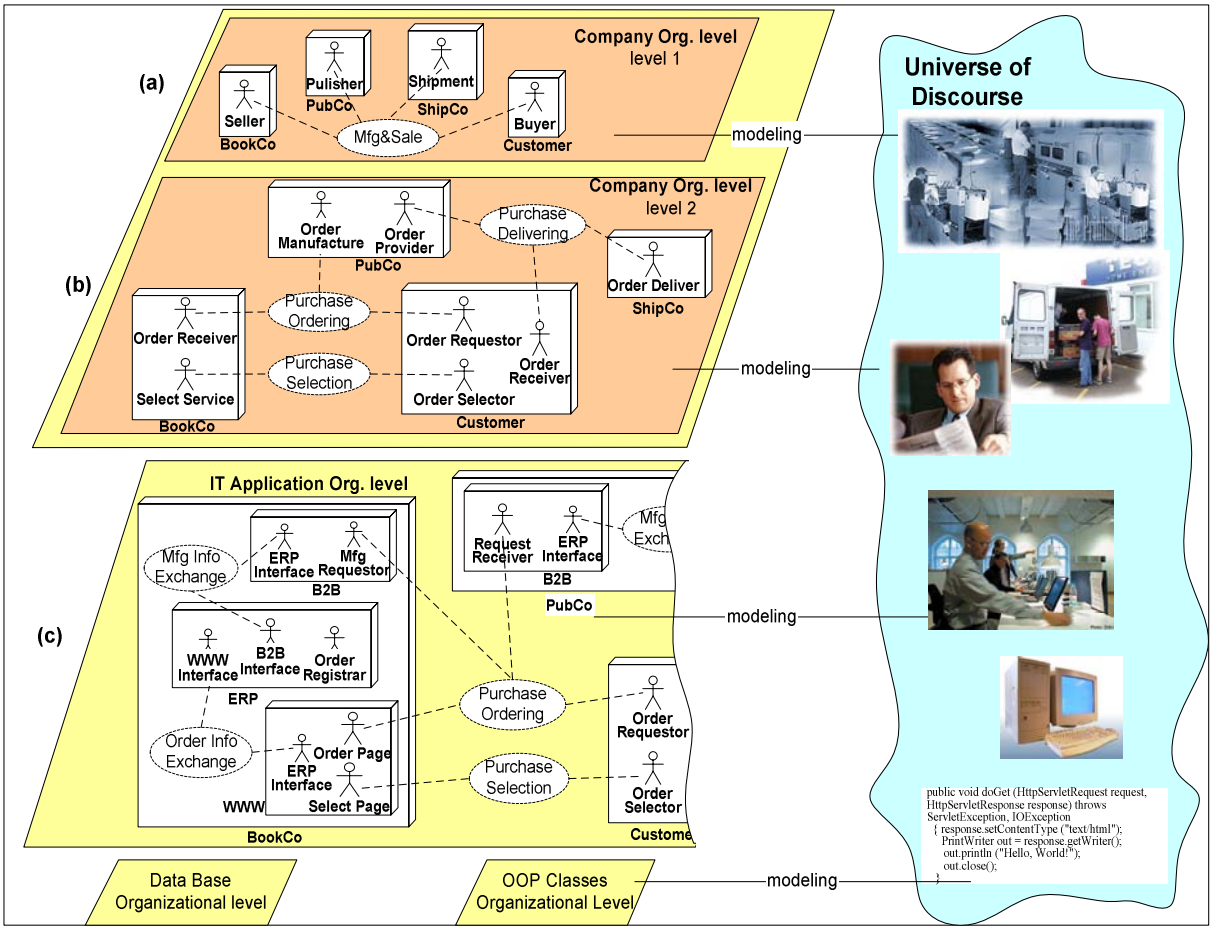


Figure 1. Multilevel model of the BookCo enterprise in EA.

Let's consider the four main principles behind the BookCo model developed using SEAM:

First, to make EA models, the EA team members have to understand what is needed to represent the reality. Designing EA models consists of identifying entities in the universe of discourse and representing them in a model. The *universe of discourse (UoD)* corresponds to what is perceived as being reality by members of the EA team and *entity* is "any concrete or abstract thing of interest" [ISO96] in the UoD. Identified entities are modeled as *model elements* in a *model*. Model elements are different modeling concepts (objects, actions, roles etc).

Second, EA team members have to understand how EA models should be designed in order to make these models understandable to each specialist. Each specialist has its own perception of the reality (or the UoD) represented as an organization level, i.e. organization levels are views of different specialists. "Each organizational level is interpreted as being made up of computational objects that represent systems" [Wegmann04].

Third, Based on RM-ODP [ISO96], to every computational object one can apply two viewpoints: the information viewpoint (IV) and the computational viewpoint (CV). We have borrowed the description of these viewpoints from the Systemic Enterprise Architecture Methodology (SEAM) [Wegmann03a]:

"The *computational viewpoint* (CV) defines the system seen as a composite (white box specification)". In CV, "a computational object is composed of several computational objects described with their IV" [Wegmann04]. I.e. in CV, each object can be refined into several computational objects at the lower organization level. Let's see an example from Figure 2 that continues the example from Figure 1 and show the refinement of the BookCo object. BookCo is refined into the B2B, ERP and WWW computational objects. Quite often this

refinement is called “implementation” because it shows how the functionality of a computational object is “implemented” with the computational objects at the lower level of the organizational hierarchy.

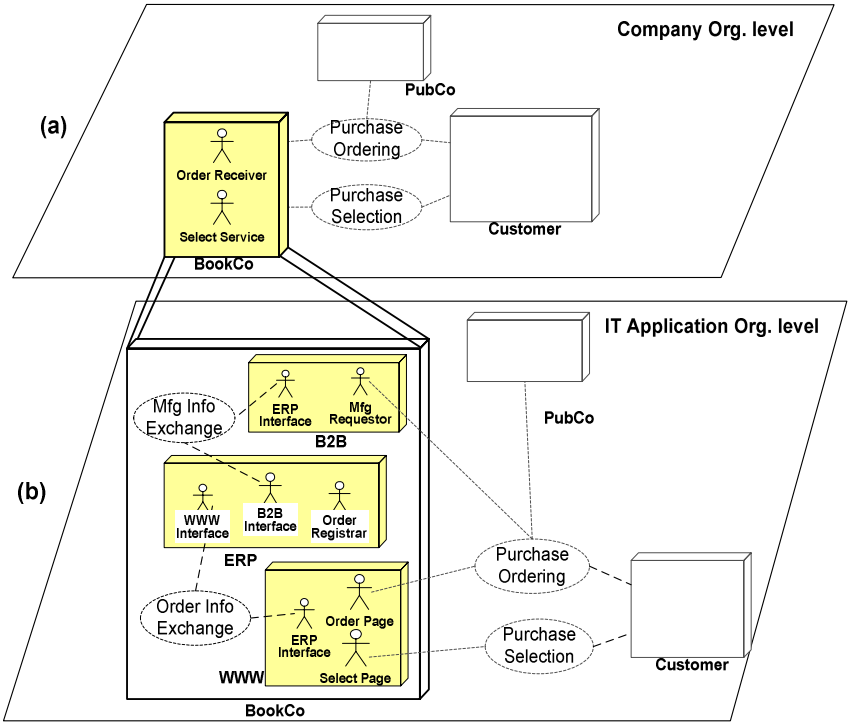


Figure 2. Connection between organization levels: the refinement of the BookCo computational object.

“The *information viewpoint* (IV) describes the system seen as a whole (black box specification). Each computational object described with their IV is specified in terms of a state and a localized action, an action that is internal to the computational object. The state of the computational object is described with concepts, called information objects that play the roles of attributes” [Wegmann04]. In other words, IV describes the functionality of computational objects. We represent this functionality by roles that an object plays in collaborations with other computational objects. For example, in Figure 1.a the information viewpoint of the BookCo object is represented by the Seller role. The functionality of a computational object can be defined at different levels of detail (or functional levels). For example, the functionality of the Seller role of the BookCo company in Figure 1.a is refined into more detailed functionality represented by the Order Receiver and Select Service roles in Figure 1.b.

The functionality and assembly of software components are two views of the same reality. The functionality provided is perceived at a higher level than the composition of components. This idea can be found in [Miller95] where he shows that the way science has developed is by analyzing the reality as a hierarchy of systems. Note, as written by [Checkland99], that this approach is also dictated by the practical need to simplify the complexity of the reality EA team members have to deal with.

IMPORTANT NOTE! The refinement and the hierarchical modeling of organization levels is a separate research topic that is addressed by other members of our laboratory[1] (see [Wegmann04] for more detailed description of the organization level hierarchy). In this thesis we are interested in modeling IV, i.e. we are interested to see how an object in its IV is specified in terms of roles.

Fourth, EA team members have to understand how to organize the management of model elements at the given organization level. As we already explained, each organization level is modeled as a set of collaborating computational objects. A *collaboration* is an action that involves multiple participants. The concept of collaboration (also called joint actions) is taken from Catalysis [D'Souza98]” [Wegmann04]. Computational objects participate in collaborations by playing roles. According to RM-ODP [ISO96], *Role* defines a subset of the behavior of an object. See for example, role A of object N in the “Do X” collaboration from Figure 3.

[1] However in order to give a better understanding of our framework we give some general suggestions in Appendix 1 where we explain how our framework is related to the refinement of the organizational levels.

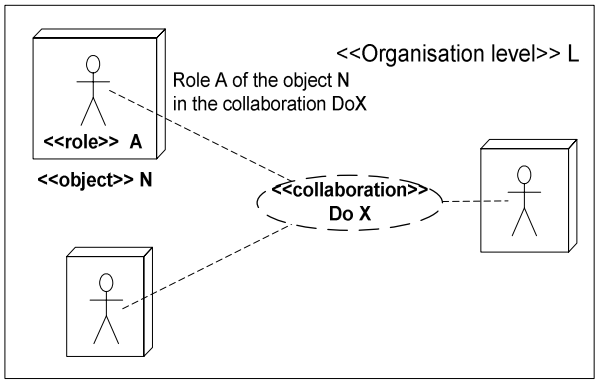


Figure 3. Concern as a collaboration between objects

Now, when we have considered principles behind EA models let's return to the question how concerns can be understood in the context of EA. Concerns can be seen as:

- Organization levels: they represent abstractions that reflect views of different specialists (modelers);
- Collaborations: they represent abstractions that reflect parts of a model at a given organization level.

In Part II of this thesis we will see more details about modeling roles and collaborations. We will introduce a detailed specification for roles that make explicit its state and behavior.

To make the state of the art analysis and the description of our modeling framework more systemic, we have used "*Systems Inquiry*" [Banathy96]. It is a discipline based on the view of the world as a system. *System Inquiry* incorporates four interrelated directions that the systemic approach uses to reason about various systems and methods: *systems philosophy*, *systems theory, systems methodology* and *systems application*. Our systemic knowledge about systems and methods is formed from the integration of the systems theory and systems philosophy, and systems methodology and systems application apply system knowledge to formulate or select methods that address real-world situations (see Figure 4).

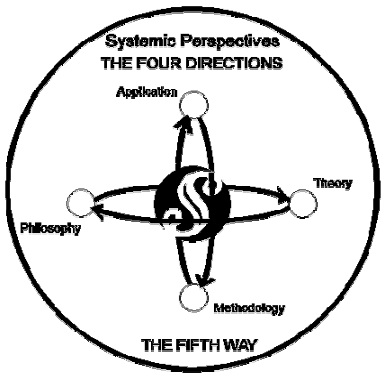


Figure 4. The four directions of the Systems Inquiry [Banathy96].

"*Systems philosophy* is concerned with a system's view on the world" [Banathy96]. This view is worked out in two directions: the *ontological* studies WHAT exists and the *epistemological* questions HOW we understand what we know. *Systems theory* tries to "recognize system properties, that are general, and structural similarities in different fields". *Systems methodology* studies different methods and "identifies specific, strategies, methods and tools appropriate to work with our system". *Systemic application* studies the application of specific models, methods and tools in some functional context.

STATE OF THE ART

Separation of concerns is widely used in software engineering: "Separation of concerns is at the core of software engineering. In its most general form, it refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose" [Ossher00]. In our work we consider Concern-Based Design Methods (CBDMs) not only on the IT level (that correspond to software engineering), but on all organization levels of EA hierarchy. In order to understand how CBDMs can be used for in EA, we have made the systemic analysis of more than twenty existing CBDMs. In this analysis we have studied how

these methods support the ontological, epistemological, theoretical and methodological principles (corresponding to the directions of System Inquiry) of concern-based modeling.

Our analysis of these methods shows that CBDMs can be divided roughly into three groups:

- **Conceptual methods**: CBDMs that are used mainly for systems analysis or requirements engineering. These methods deal with conceptual, data, business process and design patterns modeling (see, for example, Metapattern [Wisse01], Lodwick [Steimann00], ORM [Halpin01], Viewpoint UML (VUML) [Nassar03], Role Interaction Nets (RIN) [Singh92], Role Activity Diagrams (RAD) [Holt83], Role Diagrams [Riehle96]).
- **Technological methods**: CBDMs for code and data design or programming (mostly at IT organization level). In this thesis we have considered CBDMs based on architectural connectors, programming and modeling for programming methods (see, for example, CDE [Fiadeiro97], Sina [Aksit94], ConcerBASE [Kandé00], Aspect-Oriented Design [Kendall99], Stratified Architecture [Atkinson99], Aspect-Oriented Programming (AOP) [Kiczales97], Subject Oriented Programming [Harrison93], Methods for Implementing Roles [VanHilst96]).
- **Generic methods**: CBDMs that can be used for modeling most of EA organization levels, from marketing to IT implementation (see, for example, View Point Construction Framework [Nuseibeh93], OORAM [Reenskaug96], and Generic Context Framework [Mylopoulos95]).

The first two groups of methods have advantages for certain specialists, for example Aspect Oriented Programming [Kiczales97] is used by programmers and Role Activity Diagrams [Holt83] are used by business process analysts. The third group is the most appropriate for EA modeling. However, often generic methods cannot be used by all specific specialists in EA team. For example OORAM or SEAM methods are not very appropriate for programmers who work with concerns: these methods are not capable to represent explicitly all the features of concern-based programming languages (like join points in AOP). Therefore, EA projects are forced to use several CBDMs. Usually in an EA project one generic method and few conceptual/technological methods should be used.

PROBLEM

Our analysis of the aforementioned CBDMs shows that many of aforementioned CBDMs can be used in the context of EA. However, we see the following problems with using these methods in the context of EA:

- Many of the existing CBDMs support ontological and theoretical principles (defined in System Inquiry) of concern-based modeling. However, the support of epistemological and methodological principles is often missing. The study of the methodological principles goes out of the scope of this thesis. We pay attention to the epistemological principles. They play an important role in system modeling. They improve the comprehensibility of systems' models.
- CBDMs do not explain how concerns are identified and how a modeler should advance in a modeling process from the identification of base concerns to the composition of these concerns into the final specification of a system. Without a clear systemic modeling method that guides a modeler, the resulting models are difficult to understand and concerns in these models are difficult to relate one to another.

The result of our analysis shows that none[2] of the aforementioned methods solves completely both problems. Only few methods between CBDMs (such as Metapattern [Wisse01]) propose a systemic way of dealing with system concerns and using epistemological principles. However, none of these methods support all epistemological principles that we have identified in the state of the art analysis.

SOLUTION

In this thesis we solve both aforementioned problems by defining a Situation-Based Modeling Framework that is based on a set of epistemological principles. These principles were identified in our state of the art analysis. They represent the "best epistemological practices" that exist in CBDMs. These principles allows for building models that are more systemic and, therefore, more comprehensible. The following three principles we consider as the most important in the context of this work:

- Situation Relativity: a modeling method should support explicit modeling of situations.

[2] Except for SEAM [Wegmann03]. Our framework is used in SEAM.

- Data and Behavior Integrity: the data and behavior models should be integrated in the context of EA. This allows for identifying breakdowns in behavior/data interactions and for specifying explicitly the life cycle of information elements: i.e. when and where information elements are created, used and deleted.
- EA models have to be visual: this principle is especially important in the context of EA. EA design models should be used by specialists with different backgrounds. These specialists have to communicate in languages that avoid using notations specific for a certain domain.

The first principle plays the central role in the definition of our framework. In our Situation-Based Modeling Framework we propose to identify and manage system concerns by means of considering a system of interest in a number of specific situations. In our framework concerns are explicitly related to these situations. Our framework is a new CBDM where situations are the first-class modeling elements. Therefore we call our framework a situation-based one.

In order to make situations the first-class citizens in our modeling framework, we took the key principles for our framework from the situation theory. The situation theory [Barwise83] is a mathematical theory of meaning which clarifies and resolves some problems in the study of language, information, logic, philosophy, and the mind [Barwise83]. The direct application of this theory is not feasible in the context of EA because this theory is quite formal and too much adapted to linguistic needs. However, we took the basic principles from this theory as the foundation for the ontology and the theory of our framework:

- Situation-based modeling: Any entity (a system) should be considered in a number of situations. Any assertion about system properties depends on a situation.
- Explicit modeling of constraints: the mutual relations between system roles should be modeled explicitly.
- Hierarchical modeling: The behavior of an enterprise system should be represented in the form of the role hierarchy that corresponds to the hierarchy of situations.

We position our Situation-Based Modeling Framework as a generic one (i.e. it belongs to the third group of aforementioned methods) in the context of EA. Generic means that our framework can be used for modeling at different organization levels, from marketing to IT. Our framework does not prescribe any specific set of enterprise architecture deliverables. Instead it can be used as a basis to define specific artifacts that should be created at any given level of EA hierarchy. Depending on the organization level (such as marketing, business process or IT levels) our framework can be adapted for the specific needs: a modeler can define specific artifacts and define a specific method.

CONTRIBUTION

The contribution of this thesis is the original situation-based modeling framework with the accent on epistemological principles of modeling.

Comparing with other frameworks that can be used in the context of EA our framework focus on the explicit modeling of situations where a system of interest should be defined. Therefore, the novelty of this thesis is the introduction of the following principle in the context of EA:

- *Situation Relativity*: any statement about a system is relative to a certain situation. Any statement about a system can be done in the context of a certain role and can be valid only in a corresponding to this role situation.

We have made situations first-class modeling elements in the EA framework and we have made explicit the relation between system roles and *situations* in which these roles are defined. Another principle that distinguishes our framework from other CBDMs is:

- *Situation and Role inter-dependence*: Roles defined by considering systems in different situations are not independent. They usually overlap. In our work we model the mutual relations between roles with *composition constraints*. Using composition constraints allows a designer to make explicit his decisions about the composition of roles defined for different situations in an abstract and implementation independent way.

The contribution of our framework related to human-centric aspects of modeling is the integrity of data and behavior in a visual language.

- This integration allows for specifying visually the life cycle of model elements: i.e. when and where the model elements are created, used and deleted.

The main impact of this work consists in making systems' models more systemic. In our framework we make explicit different situations where the system of interest should be defined. Each model element should be

defined in a relation to one (or several) of these situations. This situation-based approach makes the definition of model elements more precise. As a result, the whole model defined as the composition of models' parts (related to the different situations) becomes more rigorous, understandable, and comfortable for reasoning.

Another important result of this thesis is a step forward in categorizing numerous CBDMs that can be used in the context of EA. Using the separation of concerns in system analysis and design has already quite a long history. First methods that used the separation of concerns (such as the early version of Object Role Modeling method [Mark87], [Halpin01] and Role Activity Diagrams [Holt83]) appeared in the seventies and the early nineties of the twenties century. Since that time a plenty of CBDMs were developed. Many of them can be used in the context of EA. The huge number of CBDMs and their diversity make it difficult to use CBDMs in EA in a coherent way. This variety of methods can cause two problems for those who develop and use innovative CBDMs in the field of EA. The first problem is to choose specific CBDMs that can be used in a given EA methodology: this is a problem for researchers who develop their own EA methodology. The second problem is to find similar methods (with the same problematic or with a similar framework) in order to make a comparative analysis with these methods: this is a problem of researchers who develop their own CBDM related to a specific problematic in EA (such as business process modeling or aspect oriented programming). To be able to answer both these questions, a set of principles should exist that can help researchers and practitioners to become orientated in the mass of the existing CBDMs.

In our framework we have defined the set of principles of concern-based analysis and design. These principles can be used:

- To evaluate, compare and choose appropriate CBDMs for using them in EA projects. In particular, generic CBDMs that can be extended with the appropriate specific CBDMs based on these principles.
- To develop new CBDMs in the context of EA or to extend existing methods with concern-based analysis and design.

OUTLINE OF THE THESIS

In this thesis we used Systems Inquiry twice to structure its contents. First, we used Systems Inquiry in Part I to classify CBDMs in the context EA:

- Part I: In this part we make the analysis of CBDMs that can be used in the context of EA. In Chapter 1 we define the *generic principles of concern-based analysis and design (CBA&D)* in the context of EA. In Chapter 2 we use the generic principles of CBA&D to generate a *list of classified CBDMs*. This list allows the researchers (i.e. designers of EA methodologies) to select the *list of relevant CBDMs for their* EA methodologies. We conclude Part 1 with a discussion about problems and trends that we have observed in the field of concern-based design and modeling in the context of EA.

The rest of our thesis closely reflects the four Systems Inquiry's directions applied to our framework:

- Part II: In this part we describe our Situation-Based Modeling Framework for Enterprise Architecture. We begin this part with Chapter 3, where we define our framework. In Section 3.1 we define the framework philosophy, in Section 3.2 we define the framework theory and in Section 3.2 we define the framework methodology. In Chapters 4 and 5 we give to examples that show how our framework is used for system modeling.

Part III: In this part we describe the application of our framework in two domains (this reflects the application of our framework): software engineering and business process modeling. In the domain of software engineering we used our framework as the basis for the Rapid Prototyping Tool based on Aspect-Oriented and Subject-Oriented programming (Chapter 6). In the domain of business process modeling we applied our framework as the theoretical background for the conceptual tool for business process innovations (Chapter 7) and for the specification of a manufacturing process (Chapter 8). The last application for the specification of a manufacturing process was done as a practical validation of our framework. We used our framework for a project in a big pharmaceutical company (see Part III of this thesis). This company needed to introduce a MES (Manufacturing Execution System) to ensure the order of the manufacturing process. The project members acknowledged the ability of our framework to improve the comprehensibility of their manufacturing process.

PART I

CONCERN-BASED DESIGN METHODS IN THE CONTEXT OF ENTERPRISE ARCHITECTURE (STATE OF THE ART)

Overview: *The goal of this part is to make an overview of the concern-based design methods (CBDMs) that can be used in the context of Enterprise Architecture (EA) and explain how these CBDMs can be used.*

In Chapter 1 we define generic requirements for CBDMs in the context of EA based on the Systems Inquiry.

In Chapter 2 we use these requirements to see how the specific CBDMs can be used to support EA modeling. We check how the requirements for CBDMs are satisfied by each CBDM. This results in the CBDM Requirements Checklist table. Then we use this table to associate each method with organization levels where the method can be used.

We conclude Part I with a discussion about problems and trends that we have observed in the field of concern-based design and modeling.

1 Definition of Requirements for CBDMs in the Context of EA Based on the "Systems Inquiry"

In this chapter, we define the requirements for concern-based design methods (CBDMs) that will be used to classify CBDMs in the context of EA. We identify the requirements for CBDMs using the four directions of Systems Inquiry (see Introduction). In Section 1.1 we consider the *systems philosophy* of CBDMs where we discuss and classify the main concepts of CBDMs. In Section 1.2 we consider the *systems theory* that defines the most general principles of CBDMs. In Section 1.3 we consider the *systems methodology* that studies the general phases (or steps) in CBDMs. In the summary of each section, we define specific Requirements for CBDMs in the context of EA. Finally in Section 1.4 we give an overview of all the requirements identified in this chapter. These requirements will be used in Chapter 2 as a basis for the classifications of CBDMs that we provide in our work.

1.1 Systems Philosophy of CBDMs

In this section, we consider the Systems Philosophy of CBDMs, which defines the main concepts and principles of the CBDMs. As we mentioned in the introduction, systems philosophy studies WHAT exists in the UoD (ontological direction) and HOW we understand WHAT we know (epistemological direction).

1.1.1 CBDMs ontology

The ontology[3] of CBDMs defines concepts that are used to describe what exists in the "reality" (entities in the UoD). CBDMs use many concepts: "view, viewpoint, role, perspective, aspect, subject, etc" (see [Nassar03]). They share many commonalities. However, as the CBDMs are developed for modeling different organization levels, these commonalities disappear behind the difference of names. In this section we consider two concepts (role and collaborations), common to all CBDMs.

In EA, each level represents a set of objects that collaborate together to fulfill some goal. An *object* is an individual component that can play roles in collaborations.

Role stands for "an abstraction of the behavior of an object" [ISO96] intended for achieving a certain common goal in collaboration with other roles. A role is usually modeled as a set of actions and state variables related to a given collaboration. In our example from Figure 1, we represented roles as stick men. Roles are usually used in EA to talk about the business or IT requirements of a system (at the Marketing, Company or IT Application Organizational levels). A concept similar to role is *feature*. This concept is also used for the specification of system requirements. Turner in [Turner98] defines feature as "a clustering of individual requirements that describe a cohesive identifiable unit of functionality". Another concept similar to role is *perspective* defined by Motschnig-Pitrik as "the data structure modeling an object relative to a context" [Motschnig-Pitrik99]. This concept is used at the Data Base Organizational level (a level below the IT Application level from Figure 1). Two concepts similar to a role at the OOP Classes Organizational level are *aspect* and *subject*. *Aspect* was first defined in AOP by Kiczales [Kiczales97] and is used for encapsulating a crosscutting code. It wraps a supplementary code and also stores information about join points and pointcuts[4]. Such join points indicate when the supplementary code should be executed. The term *subject* was proposed in the context of Subject Oriented Programming (SOP) by Harrison and Ossher (from IBM) as an extension of the object-oriented paradigm to address a problem of handling different subjective perspectives on objects to be modeled. According to [Harrison03] the term *subject* means a collection of state and behavior specifications reflecting the perception of the world. SOP uses them to represent a subjective view on objects. Any object can be seen as a composition of several subjects, where each subject can be managed separately.

Collaboration is used to describe a set of collaborating roles along with their state and behavior. For example, UML defines collaboration as a means to "describe the structural aspects and some behavioral aspects of design patterns" [OMG01]. Patterns can be considered as special collaborations "that capture the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces" [Appleton98]. The structure of patterns can be expressed as the collaboration between several objects. Patterns were introduced by Christopher Alexander [Alexander77] and became popular with the "Design Patterns: Elements of Reusable Object-Oriented Software" book [Gamma94]. A concept that is similar to

[3] We consider ontology as it is defined in computer sciences.

[4] *Join points* are well-known points in the dynamic execution of a program. And *pointcuts* are sets of join points.

collaboration is *Role Model,* introduced by T. Reenskaug. "*Role Model* specifies the set of collaborating roles along with their state and behavior" [Reenskaug96].

To make a summary of this subsection, we give the first requirement for CBDMs, which defines the basic three modeling concepts for CBDMs:

R 1.1: Requirements for CBDMs in the context of EA (Support of main ontological concepts):

- CBDMs should support the three basic modeling concepts: *objects*, *roles* and *collaborations*.

1.1.2 CBDMs epistemology

Epistemology is based on the study of human nature. It tries to understand how humans cognize. History was dominated by the ontological side of Systems Philosophy, where all attention was given to the discussion of what exists. However, from the XVII century the superiority of ontology (what exists) over the epistemology (how to explain) became questionable. Philosophers started thinking about cognitive theories but not about the Universe Existence because of a simple idea that this "world" is unlimited and can not be cognized in all its varieties. Instead, it is much easier to study the cognitive aspects of humans because they are the same for all human beings. In this case, instead of asking "what definitely exists" it is better to ask "how it is better to explain" (this analysis is based on [Gubina96]).

The first step in the epistemological shift has been taken by David Hume (1711-1776) in his "A Treatise of Human Nature". Later the epistemological shift had two schools: skeptical epistemology and critical epistemology. Skeptical epistemologists (René Descartes is the most famous member of this school) had a radical doubt about knowledge (for example, we may have a doubt about our knowledge because we can be panged into some kind of illusion that substitutes the reality). They were searching for some trustworthy elements that can be taken as a ground of knowledge. On the contrary, critical epistemologists (Immanuel Kant and Bertrand Russell are the most famous members of this school) did not believe that there was a universal ground of knowledge. Instead they used the critical approach for choosing a ground of knowledge. The critical approach reveals basic elements of knowledge, analyzes it and proves that these basic elements of knowledge can be used as a ground for the description of a phenomenon. Critical epistemologists admit the possibility of mistakes in choosing basic elements because absolute faultlessness is not achievable. With the critical approach they try to minimize chances for a mistake such that in some case these chances can be neglected.

In our work, we consider epistemology from the point of view of critical epistemologist: we introduce basic epistemological concepts and principles (we call them *epistemological Requirements for CBDMs*) that can be used as a ground for the description of EA models. We explain how these concepts and principles can improve an EA design process. For the CBDMs in the context of EA the following principles are the most important for us:

- "Explicit modeling of epistemological background principle": a model is a representation of a system from the viewpoint of a certain specialist accordingly to his epistemological background;
- "Situation relativity principle": all of human inquiry occurs within in certain situations;
- "Goal-Related principles": a goal represents the results to be achieved by a system, placed in a given situation;
- "State-Behavior Holism principle": State and behavior are inseparable in the human perception of the world: to understand state one needs to understand behavior and vice versa.
- "Principle of Visual Perception" of models: in the context of EA, design models should be simple and diagrammatic.

In the following subsections we will discuss all these principles and make a summary of each one in the form of requirements for CBDMs. These requirements we will further use to classify CBDMs.

Explicit Modeling of Epistemological Background Principle

The model of the BookCo enterprise, from our example in the introduction, is built using organization levels. These organization levels reflect the perspectives perceived by different specialists. The idea that the specialists perceive the UoD from multiple perspectives comes from social constructivism, which itself has roots in Kant's idealism. Kant "claims that we cannot know things in themselves and that knowledge of the word is possible only by imposing pre-given categories of thought in otherwise inchoate experience" (see social constructivism in [Audi99]. These categories[5] of thoughts allow EA team members to consider entities from the UoD levels, where each level explains and justifies in its own way the knowledge about a system. This explains

[5] Lakoff in [Lakoff87] explains why there are basic categories.

why EA team members have to agree, at the beginning of each EA project, on the hierarchy of organization levels. It is now necessary to explain why these views are perceived as hierarchical. At a given level a system is perceived as a whole and in another level the system is perceived as a set of its parts. The result of this part/whole relationship is the construction of a hierarchical perception of the reality. Therefore, EA models have to represent views of different specialists, where each view is used to express the attitude, perception, or opinion of an observer about a subject of interest based on his epistemological background. In the context of EA we identify two types of views.

The first type is an *organization level*. Organization levels are based on the epistemological background of different EA specialists. Many design methods (especially in the context of EA) support modeling with organization levels. For example, RM-ODP defined five viewpoints: Enterprise, Information, Computational, Engineering and Technology; Zachman framework [Zachman87] defines six architectural perspectives corresponding to the following specialists: a planner, an owner, an architect/designer, a builder, a subcontractor and a user. Note that a single organization level can be represented with multiple *levels of detail*. Levels of detail are used to make successive refinements (or adding more details) of models at the certain organization level. The refinement transforms more general models that may be valid only with certain assumptions (like non-functional requirements) to more detailed models that change assumptions to behavior. For example, the behavior of an object can be represented as a single action on the highest level of detail with some non-functional requirement. Then on lower levels of detail, this action can be refined into several smaller actions that meet the non-functional requirement. In this work, however, we disregard the levels of detail because they represent an orthogonal problem to the modeling with concerns: any concern can be modeled with one or more levels of detail and vise versa.

The second type is a *scoping view*. A scoping view can be used at any organization level and is used to extract (or filter) certain information from an existing design. A typical example here is a data-base view. In DB, views are "virtual relations being derived by a query on one or more base relations" [Motschnig-Pitrik99]. They are used to derive the necessary information from an existing data base. Another example is an actor view that is defined by means of considering concepts related with a certain actor (see for example [Nassar03]).

Note, that both views and roles (or aspects, see Section 2.1.1) are used to model system's concerns. However, they are used differently[6]. Roles, or aspects, are used as means to design a system from its parts, whereas views are used to extract information from design models.

To conclude we give the following requirement for CBDMs:

R 1.2: Requirements for CBDMs in the context of EA (Support of main epistemological concepts):

- CBDMs should support multiple *organization levels* and different *scoping views*.

Situation Relativity Principle

"History has been dominated by invariantism" [Vassallo01]. This means that only one epistemological interpretation of any model element was considered, i.e. the knowledge was considered as situation invariant. However, in the context of EA, humans need to reason about complex systems. These complex systems can be observed in different situations and interpreted differently, which means that the notion of situation allows for multiple models of the same system. Systems can be modeled differently depending on the situations in which they are. This is especially useful in the analysis of complex systems because it allows for dealing with complexity in a systematic way. In modeling the notion of context is used to represent situations in the UoD. "An essential feature is that contexts provide means to focus on aspects that are relevant in a particular situation while ignoring others" [Motschnig-Pitrik00].

The concept of context was introduced by the American pragmatist philosopher Charles Sanders Peirce in his sign model. Traditionally Peircian sign model is considered as a triad[7] <entity, model element, model element sense>: Entities (in the UoD) are modeled as model elements (in a model) with a certain model element sense (in a semantic domain), see Figure 5. However, Peirce emphasized that a model element stands for an entity, not in all respects, but in reference to a sort of idea or situation. Based on this observation, Pieter Wisse extended the Peircean sign model with the notion of context (in a model) that models a given situation (in the UoD). The result is a hexad shown in Figure 5. The original three triadic elements of Peirce reappear as dimensions in this hexad [Wisse01]. Figure 5 shows what we have just explained: a *model element* (in a model) models an *entity* (in the UoD) and has a well defined meaning: *model element sense* (in semantic domain) [see the front end triangle

[6] See details in [Champeau03], where Champeau identifies the three concern-related concepts: aspects, views, and collaborations; and discuss the difference between them.

[7] Here we use our terminology, the original Peirce triad is <object, sign, interpretant>

in Figure 5]. This model element models the entity in a given *context* (in a model). The context correspond to a *given situation* (in the UoD) and has a well defined meaning: *context sense* (in semantic domain) [see back end triangle in Figure 5].

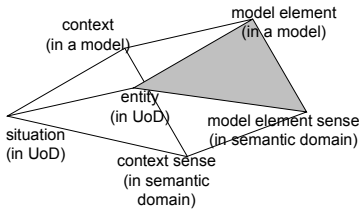


Figure 5. Pieter Wisse hexad sign model.

Figure 5 shows that the correct interpretation of a model element in a model, any model element should be specified explicitly within a context. Based on the Pieter Wisse's hexad sign model, we understand the context as the model of a meaningful situation. The situation represents what is *around* the system of interest: objects from its environment together with the joint behavior in which they are all involved. I.e. a situation can be modeled as collaboration (a set of objects in the environment of the system with their behaviors). Each role of the system of interest relates to given situation. Making situations explicit in models makes it easier to understand different roles of a system.

To conclude we give the following requirement for CBDMs:

R 1.3: Requirements for CBDMs in the context of EA (Support of main epistemological concepts):

- *Situation relativity principle*: CBDMs should support explicit modeling of situations.

Goal-Related Principle

As we mentioned in the introduction, EA models should reflect the quality of purposefulness. Goals are used by human-beings to create systems, i.e. systems are the products of their foresight and intent (teleological principle). Second, goals are used by human beings to interpret the functionality of systems: a system should have an apparent goal for a designer that explains to him the survival of this system (teleonomic principle). In the context of EA, the hierarchy of organization levels is used to model goals: at a given organization level any part of a system (or a system itself) can be considered as a whole that has a functionality with a certain result (a goal of this functionality). On the lower level of the organization hierarchy, this functionality is achieved through the collaboration of parts.

Note that goals depend on situations: a situation should have a certain consequences on an entity in this situation. Based on the System Science [Checkland99] these consequences can be modeled with goals. Therefore, a goal represents the results to be achieved by a system, placed in a given situation.

R 1.4: Requirements for CBDMs in the context of EA (Compatibility with other epistemological principles):

- *"Goal driven" principle*: A goal represents the results to be achieved by a system, placed in a given situation.

State-Behavior Holism Principle

In order to design systems, we have to describe systems' states and behaviors[8]. Many design methods and languages tend to separate systems' states from behaviors. For example, UML has separate behavior (Sequence Diagram, Activity Diagram) and state structure (Class Diagrams) diagrams related by means of contracts. The separation of state from behavior information allows for building more compact design models. However, in most interesting systems, state and behavior are highly intertwined and hard to separate. It becomes difficult to understand models of such systems when a modeler has to keep state and behavior, not related explicitly, in mind. The separation of state from behavior information goes against *Holism*, an ability to think about the system as a whole. An example of design methodology that is based on holism is the Object-Process Methodology (OPM) [Dori00]. It proposes a method for the complete integration of systems' states and behaviors within a single graphical model.

[8] We use definitions based on RM-ODP [ISO96]: *Behavior: A collection of actions and a set of (sequential) relations between actions. State: A collection of attributes, attribute values and relations between attributes.*

The holistic representation of state and behavior allows for modeling the life cycle of system attributes (or concepts). The Knowledge Management Life Cycle concept has been accepted as a crucial component of enabling KM.

This gives us the following requirement:

R 1.5: Requirements for CBDMs in the context of EA (Compatibility with other epistemological principles):

- *State-behavior holism principle*: CBDMs should not tend separate the state and behavior information of a described system.

Diagrammatic Representation Principle

This principle is important in the context of EA. EA design models should be used by specialists with different backgrounds. These specialists have to communicate in languages that avoid using notations specific only for a certain domain. For example, engineers and business analysts (or other domain experts) should have a common language. This is where the visual modeling helps: "Conceptual graphs and other diagrams have been used successfully as a communication medium between engineers and domain experts" [Sowa99]. We see the following reasons that make diagrams more convenient than textual notations for modeling EA:

- Explicit relations: Sowa in [Sowa99] explains that diagrams allow for making relations between model elements explicit. In the textual form (predicate logic, for example) relations are modeled as using variables. For example, to represent the English phrase *cats like fish,* two variables should be used in predicate logic: $(\forall x : Cat)(\forall y : Fish) like(x, y)$. In a conceptual graph these variables can be replaced by the "like" relation (See Figure 6):

Figure 6. Conceptual graph

  We can read this conceptual graph as *cats (i.e. any cat) like fish*.
- Multidimensionality of diagrams: "Multidimensionality is the essential difference between visual languages and strictly textual languages" [Burnett01]. Two or three dimensions are used in diagrams, whereas only one x-dimension is used in textual languages. One more dimension that can be used in diagrams is intensity and colors. This allows diagrams to represent multidimensional artifacts like: icons, special relationships, and non-sequential temporal (before-after) relationships. These artifacts are helpful in building models that resemble reality and therefore they facilitate model comprehension. This enables obtaining an immediate feedback on models.
- Explicit context: Sowa in [Sowa99] remarked that "the primary function of Peirce's contexts is to separate the two levels: the proposition outside the context … [and] the propositions inside". The intuitive way to specify this separation in the explicit way is to use diagrams: most of contextual diagrams use rectangles (or ovals) to make this separation. The proposition inside of contextual rectangles will be true only if the propositions outside of these rectangles are true. For example, predicates that define the functionality of a system will be true only if certain objects surround and communicate with the system of interest.
- Explicit goals: goals can be shown explicitly in diagrams by visually relating a functionality with its results. For example, one can visually relate a newly created object with an action that creates this object.
- Explicit behavior-state relation: In visual modeling a state-behavior holism principle may play an important role. Instead of the separation of state and behavior information, CBDMs may use the separation of concerns to reduce the complexity of design models. The separation of concerns based on roles, patterns and architectural views is more intuitive then a separation of systems' state from behavior.

Explicit representation of these model elements in visual languages makes modeling simpler. This allows different specialists to discuss and validate design models.

R 1.6: Requirements for CBDMs in the context of EA (Compatibility with context-related principles):

- *Principle of Diagrammatic Representation of Concerns:* Models built in the context of EA should diagrammatic. This makes the communication of larger quantity of information faster.

1.2 Systems Theory of CBDMs

The systems theory of CBDMs defines the most general properties of CBDMs. In this section we analyze these properties.

Separation of concerns is widely used in software engineering: "Separation of concerns is at the core of software engineering. In its most general form, it refers to the ability to identify, encapsulate, and manipulate

only those parts of software that are relevant to a particular concept, goal, or purpose" [Ossher00]. In our work we consider concerns concern-based methods not only on the IT level (that correspond to software engineering), but on all organization levels of EA hierarchy. When we mention CBDMs, we refer to all design methods that represent design models as sets of concerns where each concern is an abstraction that reflects a part of a system relevant to a particular concept, goal, or purpose. We believe that there are two most common forms of abstraction[9]: generalization and composition. Therefore, we consider generalization and composition as two the most common properties of all CBDMs.

We start with the definitions of generalization and composition that can be used in the context of EA. Then we continue with the identity relation that plays an important role of the composition of design parts.

1.2.1 Generalization relationship

The first form of generalization was proposed by Aristotle who introduced categories to structure the UoD. In our work we will understand generalization in the following way:

- *Generalization* is a form of abstraction that allows a modeler to classify modeling elements in terms of types and subtypes, where a type is "a predicate characterizing a collection of <X>s" [ISO96]. A subtype: "a type A is a subtype of a type B, if every <X> which satisfies A also satisfies B" [ISO96].

Using types (and subtypes) allows a modeler to be more flexible with concern-based specifications. For example, the requirement for a CBDM that an object should be able to play the same role several times can be formulated clearly using types and instances: an instance of an object type should be able to play several instances of the same role type.

R 2.1: Requirements for CBDMs in the context of EA (Support of generalization):

- CBDMs should support *generalization.*

1.2.2 Composition relationship

Today composition is studied as a distinct philosophic theory, called mereology (from the Greek μερος, 'part'). "Its roots can be traced back to the early days of philosophy, beginning with the pre-Socratic atomists and continuing throughout the writings of Plato (especially the *Parmenides* and the *Thaetetus*), Aristotle (especially the *Metaphysics*, but also the *Physics*, the *Topics*, and *De partibus animalium*), and Boethius (especially *In Ciceronis Topica*)" [Varzi03]. Today mereology gives a basis for the definition of composition in many different fields, from psychology to mathematics.

In our work we look at composition from a specific point of view: we consider how composition can be used in design methods. There are several influential works[10] that consider composition in the context of design methods:

Peter Gerstl in [Gerstl96] considers composition through the *partonomy* of its parts. A partonomy is a hierarchy of part relations. Partonomy can be based on the compositional structure of parts and properties of parts. A compositional structure defines the order of parts in a whole. For example, a house is a composition of roof, walls etc. The compositional structure in this case defines special and functional relations between house elements. This gives a partonomy of a house based on the compositional structure. If the compositional structure is not defined than a whole is only a set of parts (like a set of trees). Another Gerstl's partonomy is based on properties of parts of a whole. We can see a house as a composition of wooden, metallic or concrete parts.

Frank Barbier in [Barbier03] distinguishes between primary and secondary properties of composition. The primary properties include: *emergent property*, *resultant property* and *asymmetry*. Emergence is often is considered as a property of composition. Based on [Kilov99], a whole has to possess at least one property (emergent property) that is independent of the properties of parts. Similarly, a whole has to possess at least one property (resultant property) determined by properties of parts. Asymmetry means that a part can not have a whole as a member, i.e. asymmetry means directedness. Based on Barbier, there are quite a few secondary properties of composition. They are used to classify different types of composition. We do not consider these properties because we are interested in the most general properties of composition.

[9] Friedrich Steimann in [Steimann03] considers three main forms of abstraction (composition, classification and generalization) generalization and composition. However, in our work we consider classification as the result of generalization: instances become classified based on defined types and subtypes.

[10] Here we consider only works that aim to define composition in general, applicable to any design methodology or a modeling language.

Renate Motschnig-Pitrik in [Motschnig-Pitrik99a] considers the following properties of composition: *directedness* (the same as asymmetry from Frank Barbier), *partonomy* (or a hierarchy of parts resulting in levels of composition) and *abstraction* used to simplify design models). Similarly to Frank Barbier, Renate Motschnig-Pitrik considers some properties that can be used to classify composition.

Friedrich Steimann [Steimann03] considers composition as "the most complete form of abstraction" used in OOAD. Between the three OOAD forms of abstraction (composition, classification and generalization), only composition possess all the properties of abstractions: s*ystematic, invertible, recursive, encapsulating, structure-preserving*.

The diversity of properties of composition that we can see in the upper mentioned works can be explained by different backgrounds of authors aiming to define the primary properties of composition. Peter Gerstl, for example, mainly considers composition of physical objects. Frank Barbier and Renate Motschnig-Pitrik look at composition as conceptual modelers of IT systems. Friedrich Steimann considers composition in the context of UML models.

Our goal is to find few properties between all upper mentioned ones (they are too numerous and some of them overlap) that will be useful to define composition in the context of EA. These properties should be applicable on all levels of the EA hierarchy, i.e. we need properties that can be used by specialists coming from different domains. Therefore, we use psychology that considers how humans perceive composition in general.

In psychology there are two movements that consider the perception of composition differently: *structuralism* and *Gestalt*. The first is based on "ATOMISM – the claim that perceptual experience is the sum of numerous indivisible sensory primitives – and ASSOCIATIONISM – the claim that the glue which binds these primitives is nothing more than associative processes in the brain" [Opie99]. Gestalt theorists rejected both these claims, and instead they advocated the following: HOLISM – the claim that a perceptual whole is not reducible to the sum of its parts. Note that Holism can be explained by social constructivism. It claims that the world is accessible to us only through our interpretations. Therefore, the composition of parts in constructivism can be interpreted differently depending on the experience of each individual. This means that composition is not simply the sum of parts but also a certain properties or functionalities that an individual each individual perceives in the composed object.

Let's take an example that shows the difference between these two composition perceptions (see Figure 7) accordingly to the two movements (structuralism and Gestalt).

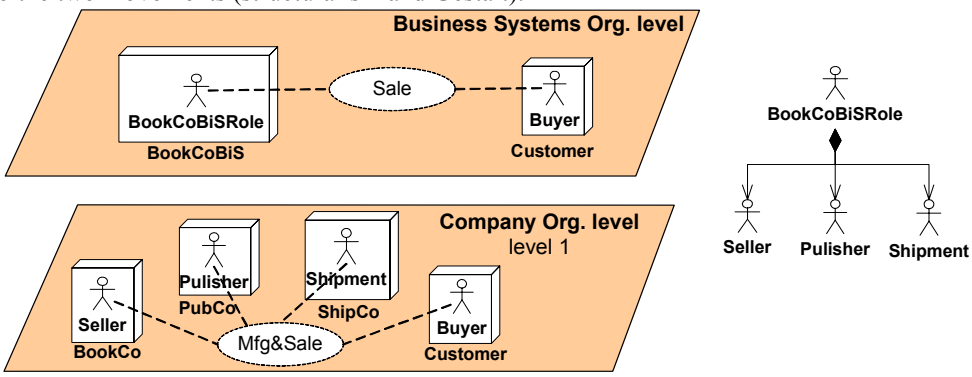


Figure 7. BookCoBiSRole as a composition of the three roles: Seller, Publisher and Shipment

In this example, we consider a new level that is one level above of the Company Organization level. This level includes the "Sale" collaboration between a Customer and a Book Company Business System (BookCoBiS). BookCoBiS plays the role of the BookCoBiSRole in the "Sale" collaboration. This role is composed from the three roles (Seller, Publisher and Shipment) roles from the Company Organizational level. How is the composed BookCoBiSRole understood according to two perceptions of composition?

Based on the structuralism, the BookCoBiSRole should be specified as the three atomic roles (ATOMISM) with corresponding associations (ASSOCIATIONISM) between them (see Figure 8.a).

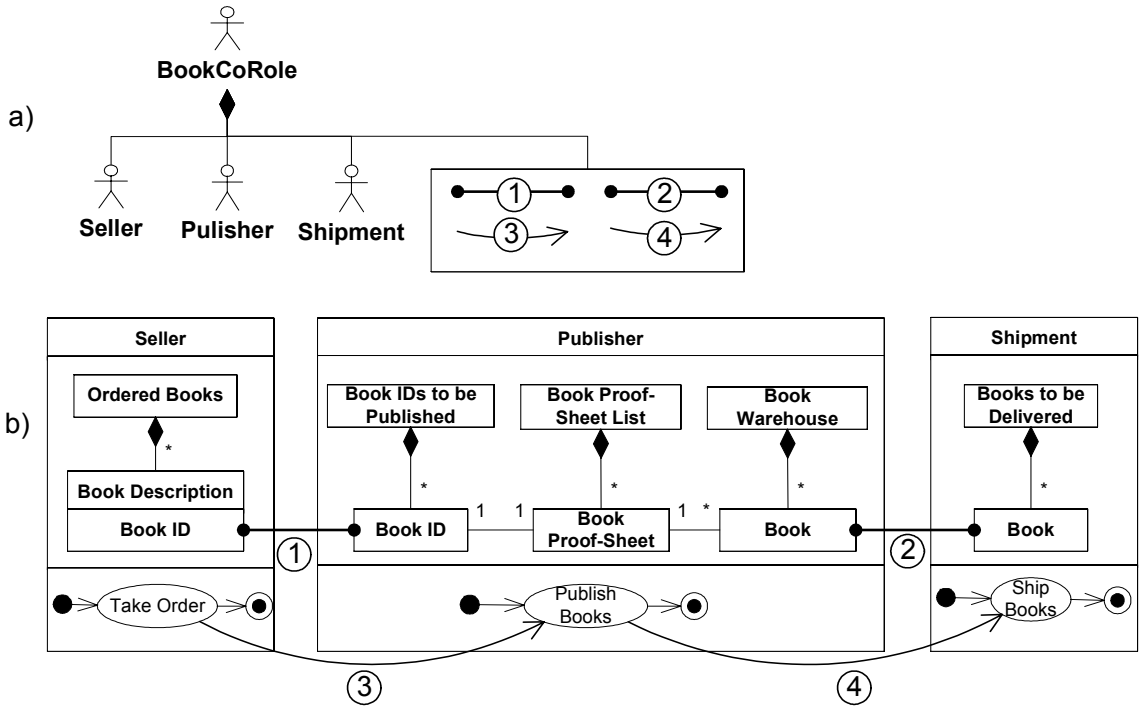


Figure 8. Composition from the point of view of the structuralism movement: a) three atomic roles; b) association between atomic roles.

How the associations from Figure 8.a are used in the composition of roles is seen in Figure 8.b. These associations represent *identity constraints* (we discuss the details about identity constraints in the next subsection) and constraints of sequentiality. In this example with the first identity constraint, we represent a fact that the Publisher manufactures books using Book Proof-Sheets. BookIDs of these Book Proof-Sheets are identical with BookIDs of the Seller, i.e. the Publisher manufactures books that that were sold to a Customer by a Seller. With the second identity constraint, we represent a fact that the Shipment role is responsible for shipping books that where produced by the Publisher. The constraints of sequentiality specify the order of actions in the composed roles.

Based on the Gestalt HOLISM, the BookCoBiSRole should be specified as a perceptual whole (Figure 9).

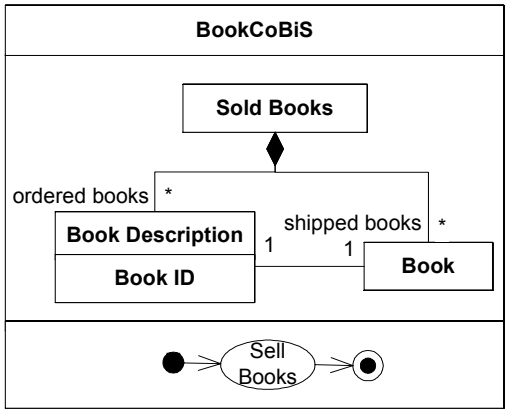


Figure 9. Composition from the point of view of Gestalt movement (Customer view): the "Sell Book" action is a new property of the BookCoBiS Role that emerges from the composition of the three base roles.

A model from Figure 9 specifies the BookCoBis company from the point of view of a customer. From this point of view the company's properties can be specified with only one action: "Sell Books". This action can be specified with the following pre-conditions: "a list of ordered books is received from a customer"; and the following post-conditions: "the set of books is shipped to a customer". Note that the set of shipped books should correspond to the set of ordered books. We specify it with one to one relation between the "Book Description" and "Book" concepts.

Gestalt movement states that the whole is observer-dependant (it depends on the perception abilities of the nervous system of an observer); and that the observer perceives the properties of the whole that in general can

not be reduced to the properties of its parts. These properties are called *emergent*. The example from Figure 9 shows the perception of the BookCoBiSRole from the point of view of a Customer. The "Sell Book" action is a new property of the BookCoBiSRole that emerges from the composition of the three base roles. Note, that the example from Figure 9 cannot be derived from base roles and associations between them (i.e. from Figure 8): in Figure 9 the Customer is not interested in the way his order is processed. He is only interested in putting an order and in receiving shipped books. To improve customer's satisfaction, we can consider a model of the BookCoBiSRole that reflects the state of the order (like "ordered", "published", shipped) that can be seen by the Customer. Therefore, attributes of the Gestalt whole depends entirely upon the choice of observer.

The both views (structuralism and Gestalt) on composition are useful in the context of EA. These both views are reflected in the following three requirements that characterize the whole as the composition of its parts (see details in [Rescher55] or in [Schünemann01a]):

R 2.2: Requirements for CBDMs in the context of EA (Compatibility with composition principles):

- *Part relationships:* "the parts of the whole must stand in some special and characteristic relation of dependence with one another; they must satisfy some special condition in virtue of their status as parts of a whole" [Rescher55]; This requirement reflects the structuralism view on composition: a whole is derived from its parts.
- *Composition structure:* "the whole must possess some kind of structure in virtue of which certain specifically structural characteristics pertain to it". The " … structural features of wholes are of interest because one important idea covered by the term `whole' is that of a structured organization of elements. A structured whole in this sense involves three things: (1) its *parts* [G], (2) a *domain of `positions'* [X] which these parts `occupy' (this need not necessarily be spatial or temporal, but may have any kind of topological structure whatever), and (3) an *assignment* [f] specifying which part occupies each of the positions of the domain" [Rescher55]. This requirement also reflects the structuralism view on composition: it defines how the whole is composed from its atomic parts. In our example from Figure 8.b, the compositional structure of BookCoRole includes the three positions of sequentially ordered roles. The first position is taken by the Seller Role, the second one by the Publisher Role and the third one by the Shipment role.
- *Emergence:* "the whole must possess some attribute in virtue of its status as a whole, an attribute peculiar to it and characteristic of it as a whole" [Rescher55]; This reflects the Gestalt view on composition that attributes of wholes cannot be derived from attributes of parts. The similar idea can be found in [Kilov99]: "There exists also at least one property of a composite instance independent of the properties of its component instances".

1.2.3 Identity

The "part relationship" requirement for composition (see the previous subsection) states that parts of a whole are mutually dependant. This dependence is specified by relations between parts. In the context of EA, there is a dependence relation that is especially important: an *identity relation*. In its classical form, *identity* is a relation between two elements (in the form: "$x=y \leftrightarrow \forall F\ (Fa \rightarrow Fb)$"). This means that the identity of *a* and *b* is implied by their sharing of all their properties. Note, if a and b are identical relative to one property (predicate), but not to another then this relation is called *relative identity*. If we read the definition of identity from right to left we can see that if two elements are indiscernible, then this means that two elements are identical. The indiscernibility of two elements can be explained by the fact that these two elements model the same entity in the UoD. Based on this observation there is another definition of identity that states: two elements are identical if they reference the same entity in the UoD. In linguistics there is a similar concept called *coreference*[11] that expresses the relation between signs.

In practice, in modeling activities "identity is often said to be a relation each thing [object] bears to itself and not to other thing" [Deutsch02]. This definition considers an absolute identity, i.e. each object has an absolute context-independent identity based on its universal self-similarity. However, the absolute identity may cause different problems. A classical example here is with two persons/passengers: "the same person might be two different passengers, since one person may be counted twice as a passenger" [Deutsch02]. For example, the Travel Co. company can count the same person (a client of this company) as two different passengers because this person travels with two different transports (a plane and a train) and requires two different insurance polices (one for each transport) (see Figure 10). The identity in this example is not absolute but relative. First, x and y are seen as identical model elements and represent a person who pays for his trip. Second, x and y are seen as

[11] Coreference is the reference in one expression to the same referent in another expression (an extract from the LinguaLinks Library [LinguaLink03]).

different model elements and represent two passengers with the different insurance IDs. This figure expresses the following statements: "x is the same person as y", but "x is not the same passenger as y". "Statements of the form "*x=y*" are incomplete therefore ill-formed. A proper identity statement has the form "x is the same as F as y" [Rea04].

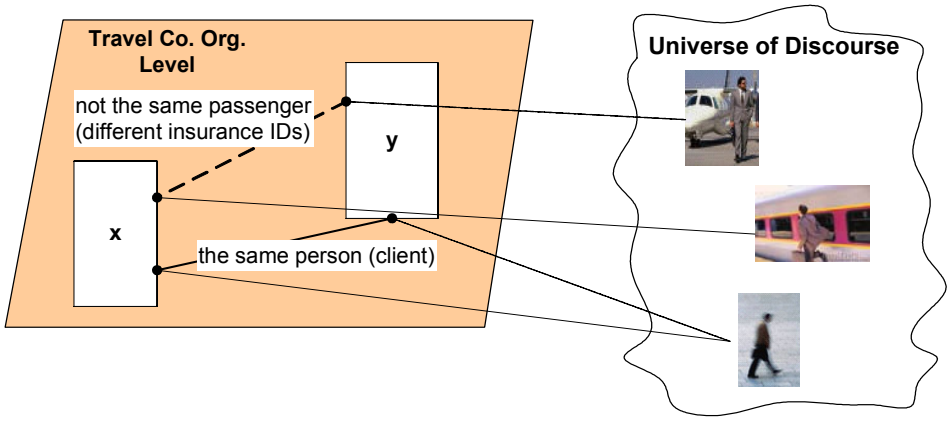


Figure 10. Identities in EA: "person identity" is a relative identity on the Person World Organization level

In the context of EA model element identities can be expressed not only on the given organization level like in the example from Figure 10 but also between organization levels. Sometimes these cross-level identities are called traces. Cross-level identities can be identified by an enterprise expert who observes several organization levels and can recognize if objects coreference the same entities in the UoD or not.

As a result we choose the following Requirements for CBDMs:

R 2.3: Requirements for CBDMs in the context of EA (compatibility with identity principles):

- *Explicit identity modeling:* In the system design using CBDMs, identity relationship between objects should be specified explicitly in the composition of concerns.
- *Explicit traceability:* CBDMs that support modeling of several organization levels, should support the cross-level identities or traceability.
- *Support of relative identity:* In the context of EA "relative identity" should be used. This means that identities should be specified in the form: "x is the same as F as y", where F is a predicate defined at a certain organization level.

1.3 Systems Methodology of CBDMs

Systems methodology identifies and studies specific methods and tools appropriate in a given context. In our work we study specific CBDMs in the context of EA. In EA an enterprise is represented as the composition of multiple organization levels reflecting views of different specialists. Any organization level may use CBDMs specific for this level. Therefore almost any CBDM can be used in the context of EA (at a certain organization level). We give the list of all CBDMs considered in our work in the next section, where we consider how these methods can be used in the context of EA.

Instead of identifying specific CBDMs that can be used in the context of EA, in this section we draw attention to some important properties of the EA methodologies. An EA methodology includes the three main development activities that should be used for each organization level (see Figure 11):

- Multi-level modeling: identifying existing entities (such as processes, goals and tools) in the universe of discourse and representing them in the multi-level enterprise model (AS-IS model). Each organization level represents entities interesting for a given specialist.
- Multi-level design: modifying the AS-IS model to fill the gap between what exists and what should exist to achieve a certain goal of a project. The result of modification is a TO-BE model on each organization level.
- Multi-level deployment: transforming the TO-BE model into new or modified entities (processes and tools) in the UoD.

The three development activities can be repeated several times. When the first iteration is finished (i.e. the deployment has been completed), then the TO-BE model takes the place of a new AS-IS model, the design and deployment are done in the second iteration and so forth.

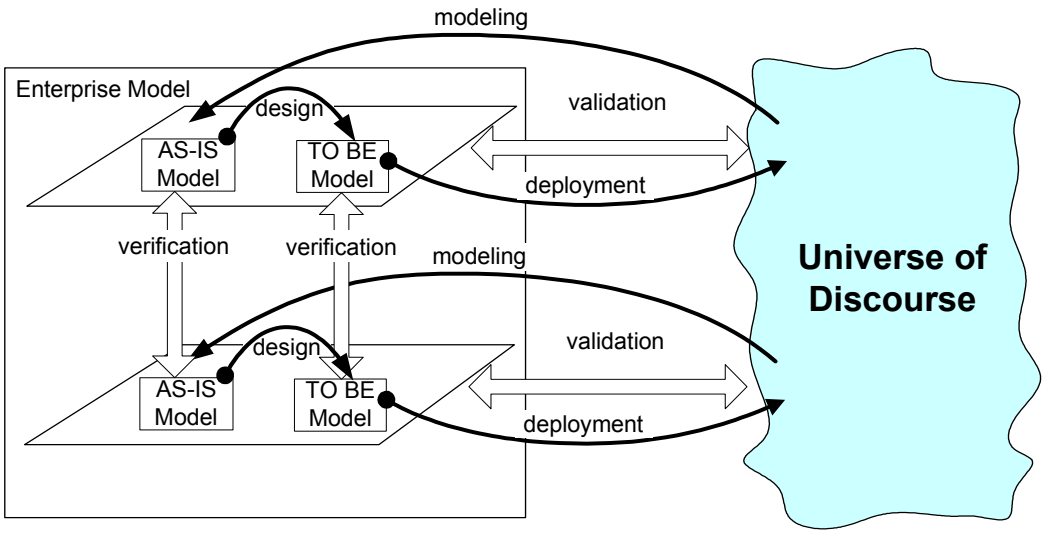


Figure 11. The role of verification and validation in the three basic development activities (modeling, design and deployment)

The result of the three development activities is available only after the model deployment (when a model becomes executable). In many cases, checking AS_IS and TO-BE models before the deployment can save the time and resources of an enterprise. There are two processes that are used to check the models: *system design validation* and *system design verification (*see Figure 11).

1.3.1 System Design Validation

The idea of the system design validation can be explained with the following phrase: build the right system. More formally, system design validation is "the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model." [AIAA98]. Design models have to be validated for each level of the enterprise hierarchy. At the given level of the enterprise hierarchy a model of a system can be defined as the composition of sub-models (defined for different situations). In this case each sub-model can be validated separately and then their composition should also be validated. This process can be quite complex since the composed models can have contradictions. In Chapter 2 we show an example how a model of system considered in different situations can be validated using axiomatic semantics.

Consistency checking and the model simulation are the most common formal techniques that can be used for model validation.

Consistency checking guarantees that a model does not have contradictions. Using consistency checking a modeler can be sure that a model (AS-IS or TO-BE) on a given organization level can be interpreted, i.e. there exists at least one scenario of the system behavior that can be interpreted in the UoD. Note, that this interpretation may be incorrect.

Simulation allows a modeler to consider specific scenarios and interpret them: these scenarios are used by a modeler to compare them with the expected behavior of the system. In result a modeler makes the conclusion about the model correctness. Simulation can be used on all organization levels. Depending of the organization level, the simulation is based on modeling elements defined at this level. For example, on the marketing level a modeler is mainly interested in the simulation of flows of goods (or services) and money between market players. On the business process level a system designer is mainly interested in the order of actions and responsibilities of business objects to perform these actions. On the IT level a system designer is mainly interested in the flow of information between IT systems.

1.3.2 System Design Verification

The idea of the system design verification can be explained with the following phrase: build the system right. In the context of EA, we will use the meaning of the system design verification taken from the Glossary of Communications, Computer, Data, and Information Security Terms [NIA2002]. It defines verification as "the process of comparing two levels of system specification for proper correspondence (e.g., security policy model with top-level specification, top-level specification with source code, or source code with object code)". We use this definition because it has a clear meaning in the EA architecture: it is defined as a process that checks if lower organization levels are consistent with higher organization levels. This means a model (AS-IS or TO-BE)

on a given organization level should be consistent with corresponding models on higher and lower organization levels (see Figure 11). As in the case of system design validation, consistency checking and the model simulation are formal techniques used for system design verification.

In the same way as for system design validation, consistency checking guarantees that a model does not have contradictions. The simulation is used differently: it is used to simulate the behavior of a system from one organization level on the other organization level.

1.3.3 Execution of the Deployed Model

This only relates to cases when the result of the deployment is formal enough to be executable. In the most of cases this is related to programming languages that allows for compiling and executing models.

This gives us the following methodological requirements:

R 3.1: Requirements for CBDMs in the context of EA (Support of the method properties):

- CBDMs in the context of EA should support system design *verification and validation.*
- The result of the deployment should be formal enough to be executable.

1.4 Summary of the Requirements for CBDMs

In this section we summarize of all the requirements for CBDMs that we have identified in the context of EA using the Systemic Inquiry (see Table 1). These requirements will be used in the next section as a basis of two classifications of CBDMs, which we provide in our work. Note, that we also add the systems application field in the end of the table that determines how a specific CBDM can be used in the context of EA, i.e. what are the EA organization levels where a given method can be useful.

Table 1. Requirements for CBDMs in the context of EA.

System Inquiry	Requirements		Name of concepts, properties or principles	Description
	#	Name		
System Philosophy	R 1.1	Support of main ontological concepts	Object	The individual, logical component that can play roles in collaborations.
			Role	An abstraction of the behavior of an object intended for achieving a certain common goal in collaboration with other roles
			Collaboration	A set of collaborating roles along with their state and behavior
	R 1.2	Support of main epistemological concepts	Organization levels	Views of different specialist that collaborate in an EA project.
			Scoping views	Used to extract (or filter) certain information from an existing design.
	R 1.3	Support of main epistemological concepts	Situation relativity principle	CBDMs should support explicit modeling of situations.
	R 1.4	Compatibility with other Epistemological Principles	"Goal driven" principle	A goal represents the results to be achieved by a system, placed in a given situation.
	R 1.5	Compatibility with other Epistemological Principles	State-Behavior Holism Principle	CBDMs should not tend to separate the state and behavior information of a described system
	R 1.6	Compatibility with other Epistemological Principles	Principle of Diagrammatic Representation	Models build in the context of EA should be simple and diagrammatic.
System Theory	R 2.1	Support of Generalization	Generalization	A form of abstraction that allows a modeler to classify modeling elements in terms of types and subtypes
	R 2.2	Compatibility with Composition principles	Emergence property	"There exists also at least one property of a composite instance independent of the properties of its component instances".
			Part relationships	"The parts of the whole must stand in some special and characteristic relation of dependence

				with one another; they must satisfy some special condition in virtue of their status as parts of a whole".
			Composition Structure	"The whole must possess some kind of structure in virtue of which certain specifically structural characteristics pertain to it".
	R 2.3	Compatibility with identity principles	Explicit identity modeling	Identity relationship between objects should be specified explicitly in the composition of concerns.
			Explicit traceability	CBDMs that support modeling of several organization levels, should support the cross-level identities or traceability.
			Support of relative identity	Relative identity should be used.
System Methodology	R 3.1	Support of the method properties	System Design Verification	Comparing two levels of system specification for proper correspondence.
			System Design Validation	Determining how well the model describes the actual behavior of the system in the UoD.
			Executable	The possibility to execute the deployed model.
System Application	R4.1	System Application	Application to EA	For each CBDM it shows EA organization levels where this CBDM can be applied.

2 Classification of CBDMs

In this chapter, we consider how the existing CBDMs support the requirements that we have identified in Chapter 1.

There are dozens of different CBDMs that can be used in the context of EA: from very generic methods to specific methods for business modeling or IT implementations. This variety of methods can cause two problems for those who develop and use innovative CBDMs in the field of EA. The first problem is to choose specific CBDMs that can be used in a given EA methodology: this is a problem for researchers who develop their own EA methodology. The second problem is to find similar methods (with the same problem domain or with similar frameworks) in order to make a comparative analysis with these methods: this is a problem of researchers who develop their own CBDMs related to a specific problem domain in EA (such as business process modeling or aspect oriented programming). In this chapter we address both of these problems by means of the classification of some influential CBDMs in the context of EA.

2.1 List of Concern-Based Design Methods

We begin this chapter with the description of each method from the list of CBDMs that we considered in our work. This list of CBDMs in this section is not exhaustive, but we included several influential methods in each application domain. The CBDM application domains represent our understanding of the positioning and goals of CBDMs as presented by their authors. We use the following application domains: generic methods, DB and conceptual modeling methods, business process related methods, design pattern modeling methods, methods based on architectural connectors, programming and modeling for programming methods.

For each method we describe its goal, main concepts and principles. We also show how these concepts and principles relate to the requirements for CBDMs from Chapter 1 of this thesis.

Generic Methods:

Generic CBDMs use general terms to describe the deliverables of EA. These methods usually cover several EA organization levels. Generic CBDMs can be used together with specific CBDMs: in this case generic deliverables at a given organization level can be substituted with specific deliverables, more adequate for a certain specialist. We have considered the following generic CBDMs:

- **Object-Oriented role analysis and modeling OORam:** a generic method developed by **T. Reenskaug**
  Goal: Modeling complex systems as structures of interacting objects.
  Concepts and Principles: *object* (r1.1), *role* (r1.1), *role model* (or collaboration, r1.1), *composition of role models* (or composition) based on generalization of behavior (r2.1), behavior substitution (or behavior identity, r2.3) and sequential composition (part relationship r2.2).
  Description: OORam is a highly successful generic methodology applied mostly in software engineering and with some opportunities for business analysis [Reenskaug96]. This is the first work that systematically studied and used the concept of role, role model and composition of role models. OORam pays attention to the *safe composition* that guaranties the integrity of the base model activities in the derived models.
  Tool: "OOram Professional" was developed. However, this case tool is not supported any more.
  Advantages: Simple, visual (r.1.6), encourages model and software reuse.
- **Systemic Enterprise Architecture Methodology (SEAM):** a generic method developed by **A. Wegmann**
  Goal: Systemic hierarchical system modeling in the context of EA.
  Concepts and Principles: *organization level* (r1.2), *object* (r1.1), *role* or localized action (r1.1), *joint action* (or collaboration, r1.1), the explicit modeling of context and environment (r1.3), visual representation of post-conditions (state-behavior holism, r1.5).
  Description: SEAM [Wegmann03] is a new generic method in the field of EA. This method has solid theoretical foundations and aims to be used by practitioners in the field of EA. It supports many properties that we have identified in Section 2.
  Tool: SEAMCad (see http://lamspeople.epfl.ch/le/SEAMtool/SeamCad1/toc.htm) is under development.
  Advantages: Visual (r.1.6), systemic, good support for the hierarchical modeling, proposes a unified otology for all EA levels and, therefore, can be used as basis for the development of the specific EA methods.
- **ViewPoints Framework:** a generic framework proposed by **B. Nuseibeh** et al.
  Goal: To make explicit inter-viewpoint relations.
  Concepts and Principles: *viewpoint* (organization level, r1.2), viewpoint *relationship rules* (part relationships, r2.2), *agent* (object, r1.1), correspondence between types in different viewpoints (explicit traceability, r2.3).

Description: ViewPoints Framework is a viewpoint method construction framework that defines how to construct methods that integrate multiple viewpoints. It suggests how to create viewpoint templates and use these templates for building hierarchical specifications. This framework addressed "the notion of inter-Viewpoint communication as a vehicle for Viewpoint integration" [Nuseibeh93]. Inter-viewpoint communications are based on the inter-viewpoint rules defined in viewpoint templates. The paper [Nuseibeh93] that describes the basics of ViewPoints framework in 2003 got the Most Influential Paper Award in ICSE, one of the leading conferences in Software Engineering.

Tool: The direct successor of the ViewPoints framework is xlinkit framework [Nentwich03] (see http://www.xlinkit.com) that aims in consistency checking and repairing of software engineering artifacts. The xlinkit framework has a case tool of the same name.

Advantages: Helps to make relations between viewpoints more formal and, therefore, allows for the consistency checking between viewpoints.

- **Generic Context Framework**: a generic framework proposed by **R. Motschnig-Pitrik**

  Goal: To decompose Information Bases (IBs) using contexts.

  Concepts and Principles: information units (all possible model elements such as objects, attributes, methods)*; context* (something like a role in our terminology, r1.1) refers a subset of information units, i.e. context is used to split IBs; *authorization* (specify user rights for the execution of actions), *change propagation* (specify part-relationships between contexts, r2.2) and *owner* (a user who create a context).

  Description: This framework originally proposed in [Mylopoulos95] is inspired by data base views. It considers general abstractions for partitioning information bases with contexts. Later this framework was presented as a generic one for any modeling notation [Motschnig00a]. It proposes a context to be a "first-class citizen" associated with properties and behavior of objects. This framework was applied in the context of OO modeling, which resulted in the extension to UML that supports the modeling of views [Motschnig00b].

  Advantages: This generic framework is strongly influenced by the data base views and, therefore, it is most appropriate for the development of IBs specific frameworks.

DB and Conceptual Modeling:

A group of methods that make an accent on the information modeling for information systems.

- **Lodwick:** a modeling language proposed by **Friedrich Steimann**

  Goal: To define the semantics of roles in conceptual modeling.

  Concepts and Principles: object (r1.1); *role* (r1.1); *role and object type hierarchies* (generalization, r2.1); *static model* or invariant model; *dynamic model* that specifies all possible sequences of model snapshots.

  Description: Lodwick is one out of few languages based on logic (order-sorted logic more precisely). It "is intended to be an exploratory language for object-oriented modeling with roles at the conceptual level" [Steimann00].

  Advantages: Lodwick provides a good formal definition of roles and their properties that can be used to explain the semantic of roles in CBDMs. Lodwick allows for the expression of many propositions about roles such as: "roles can play roles", "a role can be transferred from one object to another", "an object my play the same role several times" etc.

- **Object-Role Modeling (ORM)**: a modeling method considered by **Halpin**

  Goal: To simplify the conceptual design by using natural language, intuitive diagrams and representing information in terms of simple or elementary facts.

  Concepts and Principles: object (r1.1); *role* (r1.1); *facts* or *n-ary predicates* (resemble to collaborations, r1.1) that can be regarded as sentences with one or more "object-holes" where each hole represents a *role*; *constraints* between roles (part relationships, r2.2); role *generalization* (r2.1); visual and simple (r.1.6)

  Description: ORM is a method for performing information analysis and design at the conceptual level. It is considered as an alternative to Entity-Relationship – a group of conceptual modeling methods. "Early versions of ORM were developed in Europe in the mid-1970s (for example, binary relationship modeling and Natural Language Information Analysis Method (NIAM))" [Halpin01]. ORM is one of the methods appropriate for EA because it "simplifies the design process by using natural language, as well as intuitive diagrams which can be populated with examples" [Halpin01].

  Tool: Microsoft Visio for Enterprise Architects; in addition, ActiveQuery can be used for querying ORM models.

  Advantages: ORM diagrams are simple and can be used by many different specialists (and especially business people). ORM models can be converted to database schemas, ER and UML diagrams.

- **Metapattern**: a conceptual modeling method proposed by **P. Wisse**

  Goal: Information analysis and design that uses context and time as first-class modeling elements.

Concepts and Principles: object (r1.1); *context* (r1.3); *intext* (or role, r1.1)*; information objects* (or role attributes); *pointer information object* (or identity relation, r2.3);
Description: Metapattern [Wisse01] uses a simple visual notation in the form of a directed graph where nodes represent objects and edges represent contextual relations between objects. Any object in Metapattern can be defined only in the context of another object using a contextual relation. The directions of contextual relations show the order of nested contexts.
Tool: KnitbITs is a commercial tool for prototyping. It assists strategic planning for enterprise engineering.
Advantages: Highly focused analysis tool that provides a formal treatment of context; a visual notation (r1.6) for the representation of roles, contexts and relations between them for modeling at the high level of abstraction.

- **VBOOL and VUML**: a methodology and modeling language proposed by **Nassar** and others
  Goal: To introduce the notion of a user view associated to every actor of a system.
  Concepts and Principles: *actor* that interact with a system (object, r1.1); *view* of an actor on a system (role, r1.1); *view dependencies* (part relationships, r2.2); view extension (specialization/generalization, r2.1).
  Description: View Based Object-Oriented Methodology Language (VBOOL) [Marcaillou94] is an OO language based on multiple inheritance (inspired by Eiffel) with the explicit notion of user views. View based Unified Modeling Language (VUML) [Nassar03] is an extension of UML that introduces user views associated to every stakeholder of a system. VUML was inspired by VBOOL.
  Tools: VBOOL interpreter.
  Advantages: Allows for the specification of user needs and access rights in a visual way (based on the extension of UML class diagrams).

Business Process (BP) related methods:

BP related methods aim at the analysis and design of workflows and processes in an organization.

- **Role Activity Diagrams (RAD)**; a visual language proposed in by **Holt** and enriched by **Ould**
  Goal: To express coordinated human behavior.
  Concepts and Principles: *role* (r1.1); *actor* (or objects, r1.1); *interaction* (or detailed collaboration, r1.1); *goal* of a role in collaboration (r1.4).
  Description: RADs are based on concepts proposed by Holt et al [Holt83]. Later RADs were improved by Ould [Ould95]. RADs were a major feature of the Business Process Reengineering movement in the 1990's. RADs are similar to UML Activity Diagrams (ADs) with swim lanes. They are different in the visual notation and the model elements that can only be modeled in RADs (goals, data flows, interactions between roles). These model elements make RADs more comfortable for business process modeling (see [Odeh02] for details on the comparison). RADs are based on the underlying Petri-Nets formalism.
  Tools: RADRunner (see http://www.rolemodellers.com) is a commercial product with the underlying XML dialect, Playwright, that allows for the integration with web technologies.
  Advantages: Allows business processes to be expressed visually (r1.6) at a high-level of abstraction. RADs are well known and supported with a powerful tool.
- **Role Interaction Nets (RINs)**; a visual language proposed by **Singh** and **Rein**
  Goal: To express coordinated human behavior.
  Concepts and Principles: *role* (r1.1); *actor* (or objects, r1.1); *interaction* (or detailed collaboration, r1.1); *output* (or goal of a role in collaboration, r1.4).
  Description: RINs [Singh92] are very similar to RADs: they both use similar concepts, principles and based on the Petri Nets formalism. However, RINs did not advance as much as RADs. As a result, RINs are not referenced in recent research publications.
  Tools: Deva [Rein93] is a role-based collaborative tool that allows people to coordinate their work. This tool does not exist anymore.
  Advantages: Allows business processes to be expressed visually (r1.6) at a high-level of abstraction.

Design pattern modeling methods

Design patterns are proven solutions to recurring problems. Patterns in the field of object-oriented analysis and design were first studied by Gamma in his book "Design Patterns: Elements of Reusable Object-Oriented Software" [Gamma94]. These patterns are usually described using class diagrams. Some researchers go beyond class diagrams and develop new languages for the specification of design patterns. Here we overview one approach, relevant to the subject of our work, that specifies patterns as a set of collaborating objects.

- **Role Diagrams:** proposed by **Dirk Riehle**

Goal: To specify design patterns as a set of collaborating objects and show how design patterns are applied to objects.
Concepts and Principles: *class*, i.e. a set of objects of a given type (r1.1); *role* (r1.1); *collaboration* (r1.1); role and class *generalization* (r2.1); *composition constraints* (r2.2); *a role diagram* represents a set of collaborating roles together with composition constraints, generalization and composition relations between roles; *a class model* shows how classes play roles from role diagrams.
Description: Role diagrams are the major contribution of the Riehle's research work. They are quite simple and powerful for solving concrete design problems. Role diagrams where inspired by OOram and the work of C. Alexander [Alexander77].
Advantages: Convenient for the description of design patterns in a visual way (r1.6) [Riehle96], [Riehle97]; convenient the design of OO Frameworks [Riehle98].

Architectural connectors

Methods based on the architectural connectors (we take this name from [Fiadero97]) aim to separate object essential behaviors (services provided by this object) and object interactions. This separation allows for the explicit representation of object interactions in the form of contracts, communication objects or connectors.

- **CDE from ATX Software:** proposed by **J. Fiadeiro, L. Andrade** at al.
  Goal: To separate basic business components from coordination elements (business rules) managed by configuration elements (business policies).
  Concepts and Principles: *components* or basic business blocks (i.e. objects, r1.1); *coordination contract* (something like collaboration, r1.1); *configuration elements* or business policies (goal of a coordination contract, r1.4); computation, coordination and configuration layers (organization levels, r1.2).
  Description: It uses coordination contracts to represent explicitly the rules that determine Java object interactions [Andrade99], [Fiadeiro97]. Coordination contracts support interactions to be externalized as first-class citizens, allowing for the online deployment of coordination contracts.
  Tool: CDE is the Java-based Coordination Development Environment (CDE). CDE also allows for the simulation of the coordination mechanism using an animation tool integrated in CDE.
  Advantages: Allows for the dynamic reconfiguration of a system caused by the changes of business policies and rules.
- **Sina**: a programming language proposed by **Mehmet Aksit** at al.
  Goal: To structure, abstract and reuse object interactions.
  Concepts and Principles: *object* (r1.1); *Abstract Communication Types, ACTs* (represent object collaborations, r1.1, or inter-object constraints, i.e. part relationship, r2.2); *composition filters* are used to intercept and redirect messages from objects to ACT objects.
  Tool: Sina [Aksit88] is a programming language with the explicit representation of object interactions in the form of Abstract Communication Types (ACTs) [Aksit94]. ACTs represent explicitly complex communications between objects, such as distributed algorithms, coordinated behavior, inter-object constraints.
  Advantages: Makes the complexity of programs manageable by moving the interaction code to separate modules; can implement the synchronization among participating objects.
- **ConcernBASE:** a language and method proposed by **M. M. Kandé**
  Goal: To provide a software engineering approach that allows for the separation of concerns in software architecture descriptions.
  Concepts and Principles: *components* (i.e. objects, r1.1); *connectors* (similar to collaborations, r1.1); connector consists of two or more *connection points* (similar to a role in our terminology, r1.1) and one *connection role* or a communication protocol between connection points.
  Description: ConcernBASE (see http://lgl.epfl.ch/research/concernbase/index.html) is a concern-based and architecture-centered software engineering method. To represent object interactions, ConcernBASE uses connectors. "A connector is an abstraction that explicitly represents a locus of definition for component interconnections and communication responsibilities" [Kandé00].
  Tool: The ConcernBASE Modeler tool is an integrated tool for the development of UML-based architectural descriptions using the Concern-BASE approach. This tool is currently under development.
  Advantages: UML-based; complements current Architecture Description Languages (ADLs) with the separation of concerns mechanism.

Programming and Modeling for Programming

The common goal of methods from this section is to implement roles as source-code entities and then assign these roles to objects.

- **Methods for Implementing Roles** proposed by **D. Notkin** and **M. VanHilst**
  Goal: To implement roles as C++ code entities and compose them into classes using separate composition statements.
  Concepts and Principles: *roles* in the form of C++ templates (r1.1); *classes* (that instantiate objects, r1.1); *composition statements* that specify how roles are composed with classes (something like composition structure, r2.2); *roles/responsibility matrix* where rows represent collaborations (r1.1) and columns represent classes.
  Description: This method implements roles as source code entities using C++ class templates defined in a stylized way [VanHilst96]. These templates are composed into C++ classes at compile time using separate composition statements. The composition statements are based on the roles/responsibility matrix that is used to define relations between roles' attributes and methods, and the order in which these roles are composed.
  Tool: C++: it uses the features associated with class templates in C++.
  Advantages: Improves C++ code maintainability and reuse; requires no special tools.
- **Subject-Oriented Programming (SOP):** proposed by **Harrison and Ossher**
  Goal: An extension of OOP that addresses a problem of handling different subjective perspectives on objects to be modeled;
  Concepts and Principles: *object* (r1.1); *subject* is a collection of object's state and behavior specifications related to a particular concern (or a role, 1.1); *concern* is an expectation or a goal that a stakeholder has on a system (can be represented as collaboration with a goal, r1.4); *composition specification* (specifies part relationships, r2.2) that includes *composition relationship* (or identity relationship, r2.3) and *integration specifications* (tells how the identical elements should be treated: merged, overwritten or selected depending on a context).
  Description: Subject Oriented Programming is a programming paradigm proposed by Harrison and Ossher (from IBM) [Harrison93]. SOP uses subjects to represent a subjective view on objects. Any object can be seen as the composition of several subjects, where each subject can be managed separately. The most known implementation of SOP is the Hyper/J language (see http://www.alphaworks.ibm.com/tech/hyperj).
  Tools: Hyper/J – a Java application that allows for the composition of conventional Java classes according to composition rules. Hyper/J composes Java class files based on the special options file. This options file indicates files that participate in a composition, how parameters or actions with the same names should be treated, and other complimentary information.
  Advantages: SOP is as generic programming paradigm that allows for the separation of concerns. The separation of concerns helps to trace requirements (in the form of use-cases or features) to code (in the form of programming concerns); SOP improves comprehensibility; SOP has an open-ended semantics of composition that allows for the definition of complex composition patterns.

- **Aspect-Oriented Programming (AOP):** programming language introduced by **Xerox**
  Goal: To localize code that is scattered across several classes.
  Concepts and Principles: *source code* (instantiates objects, r1.1); *advice* is the cross-cutting code to be added to the source code of objects (something like role, r1.1); *point-cuts* (r2.2) specify the composition structure of a source code with advices; *aspect* is the combination of point-cuts and advices.
  Description: Aspect-Oriented Programming was named by Gregor Kiczales and his group [Kiczales97]. It was based on the ideas of adaptive programming that were developed in the early 90th [Lieberherr92]. AOP paradigm introduces a new concept to OOP called *Aspect* for encapsulating a crosscutting code. There are many examples of aspects: error checking and handling, synchronization, context-sensitive behavior, performance optimizations, monitoring and logging, debugging support, multi-object protocols.
  Tools: The first version of the AOP language and language processor, AspectJ, that interleaves or weaves objects and aspects was done by Gregor Kiczales, Crista Lopes and other researchers at Xerox PARC. Now AspectJ has evolved into a powerful AOP framework. The information about other AOP frameworks can be seen on http://aosd.net/technology/practitioners.php.
  Advantages: Modularization: redundant code can be placed in aspects; Concentration on the business logic: security, synchronization and other non-business concerns can be handled with aspects; Comprehensibility; Debugging: debugging code can be outside of the main code; Acceptance in industry: integration with developer frameworks such as JBoss (JBossAOP), NetBeans and Eclipse.
- **Aspect-Oriented Design**; design method proposed by **Elizabeth A. Kendall**
  Goal: A role-based design method and its implementation in AOP.

Concepts and Principles: *classes* (that instantiate objects, r1.1); *aspects* and *roles* (r1.1); *role models* (i.e. collaborations, r1.1).
Description: E. Kendall in [Kendall99] proposes an aspect-oriented design method that can be implemented using AOP. This design method is specified with role diagrams proposed by Dirk Riehle [Riehle97]. It also uses the graphical notation of role composition inspired by [Kristensen95]. E. Kendall considers different options for mapping roles from role diagrams to aspects in AOP and discusses the advantages and problems of these options. In [Kendall98] E. Kendall discusses how goals of a system can be specified and assigned to roles that appear in a model.
Advantages: Simple visual (r1.6) notation that allows for choosing different options of role model implementations; intuitive for the implementation of design patterns with AOP.

- **Stratified Architectures**: a method and architecture proposed by **Colin Atkinson** and **Thomas Kühne**
  Goal: To provide a method for hierarchical modeling that uses concern-based abstractions.
  Concepts and Principles: *strata* or level of abstraction (r1.1); *object* (r1.1); *object interaction* (i.e. collaboration, r1.1); *interaction refinement* based on the introduction of new system concerns.
  Description: Colin Atkinson and Thomas Kühne in [Atkinson99] propose a systematic organization of concern-based models in a form of hierarchical structure. This structure allows for the abstraction of "system details step by step so that certain aspects [concerns] can be ignored at a sufficiently high level of abstraction" [Atkinson99].
  Advantages: Comprehensibility: stratified architectures make explicit why and where (at which level) a particular concern is introduced in a system, and what are the implication of this concern on the system's overall structure; System redesign becomes easier.
- **OO Modeling with roles:** a methods proposed by **Kristensen**
  Goal: Modeling of perspectives based on the aggregation of roles.
  Concepts and Principles: *perspective* (similar with organization levels, r1.2); *intrinsic object* (or object, r1.1); *role object* (or role, 1.1); *generalization* (r2.1); *emergent methods and attributes* (r.2.4); method and attribute dependencies (or part relationships, r2.2) such as hereditary, aggregated, modified methods.
  Description: Bækdal and Kristensen use roles to specify systems from different perspectives [Bækdal99], [Kristensen95]. Each perspective models a system with its own aggregation hierarchy: a perspective defines roles at the lowest level of abstraction and then aggregates them in a form of a hierarchy.
  Advantages: Simple and rich visual notation (r1.6) that allows for modeling the specialization and the composition of roles, relations between roles, assignments of roles to other roles and etc.

The following papers can also be consulted about experiments with programming languages to support roles and role modeling: [Jensen95], [Tinggard95].

We check how the aforementioned requirements for CBDMs are satisfied by each CBDM. This results in the *CBDM Requirements Check* table. Then we use this table to associate each method with organization levels where this method can be used. This results in the *classification of CBDMs in the context of EA*.

2.2 CBDM Requirements Checklist

In this section we show CBDM Requirements Checklist Table (see Table 2) where we show how the aforementioned methods satisfy requirements for CBDMs. To make a conclusion about the evolution of considered CBDMs, we sort the considered CBDMs by the date of their appearance. We associate this date with a first major publication that we have found in the literature. Note that we also included in this table the Systemic Application column that indicates where each CBDM is more appropriate: in business or IT.

Table 2. CBDM Requirements Checklist Table

Year	Method/ Language Name	Authors	Tool or Language	System Philosophy									System Theory							System Methodology			System Application
				Ontological Concepts			Epist. Concepts			Other Epist. Principles				Composition			Idenity principles						
				object	role	collaboration	Organization levels (views)	Scoping views	Situations	"Goal Driven" contexts/roles	State- Behav. Holism	diagrammatic representation	generalisation	Emergent Properties	Part relationships	Composition Structure	Explicit Identity relat.	Explicit Traceability	Relative Identity	Desing Verification	Design Validation	Executable	
approx. 1975	Object Role Modeling	Halpin	FORM in MS Visio for Enterprise Architect	Y	Y							Y	Y		Y							supports conceptual queries	IT, Business
1983	RAD	Holt at al	RAD		Y	Y			Y	Y		Y			Y								Business
1988	Sina	Aksit	Sina language	Y	Y	Y							only through delegation		Y	Y						Y	IT
1992	RINs	Singh Rein	RINs		Y	Y			Y			Y			Y						Simulation		Business
1993	SOP	William Harrison	Hyper/J	Y	Y	Y							Y	Y	Y	Y	Y					Y	Business
1993	ViewPoints Method Construction Framework	B. Nuseibeh	xlinkit				Y						Y		Y			Y		Consistency Checking	Consistency Checking		IT, Business
1994	VBOOM, VUML	Marcaillou at al.	VBOOL, VUML		Y		Y					Y	Y										IT, Business
1995	Generic Context Framework	R. Motschnig-Pitrik	extention to UML	Y	not in the same meaning				Y			Y	Y		Y		Y						IT, Business
1995	OO Modeling with Roles	B. Kristensen		Y	Y	Y	Y					Y	Y	Y			Y						IT
1995	OORam	Reenskaug	TASKON/ OOram	Y	Y	Y	Y		Y	Y		Y	only between modules		Y	Y		Y					IT, Business
1996	AOP	Kiczales	AspectJ	Y	Y	Y							Y			Y	some						IT
1996	Methods for Implementing Roles	Notkin		Y	Y	Y							Y		Y		some						IT

Year	Method/ Language Name	Authors	Tool or Language	System Philosophy									System Theory							System Methodology			Systemic Application
				Ontological Concepts			Epist. Concepts			Other Epist. Principles				Composition			Idenity principles			Desing Verification	Design Validation	Executable	
				object	role	collaboration	Organization levels (views)	Scoping views	Situations	"Goal Driven" contexts/roles	State- Behav. Holism	diagrammatic representation	generalisation	Emergent Properties	Part relationships	Composition Structure	Explicit Identity relat.	Explicit Traceability	Relative Identity				
1996	Role Diagrams	Dirk Riehle		Y	Y	Y			Y			Y	Y		Y	Y							IT, Business
1999	CDE from ATX Software	J. Fladeiro, L.Andrade at	CDE	Y	Y	Y			Y				Y								Simulation	Y	IT
1999	Aspect-Oriented Design	Elizabeth A. Kendall	based on AOP	Y	Y	Y			Y	Y		Y	Y		Y							Y	IT
1999	Stratified Architectures	C. Atkinson and T. Kühne		Y	Y	Y	detail levels		Y		Y	Y	Y					Y				based on AOP	IT
2000	ConcernBASE	Kandé	The ConcernBASE Modeler	Y	Y	Y	Y	some	Y	Y		Y											IT
2000	Lodwick	F. Steiman	Lodwick	Y	Y								Y			Y							IT, Business
2001	Metapattern	P. Wisse	KnitbITs tool-set for prototyping	Y	Y	Y	Y		Y			Y			Y	Y	Y	Y					Business
2003	SEAM	A. Wegmann	SEAM Cad Tool	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y		Y	Y	Y		Consistency Checking		IT, Business

We use Table 2 to associate each method with organization levels where this method can be used. This results in the classification of CBDMs in the context of EA (see Figure 12, next page). Methods that support multiple levels of EA are more interesting for enterprise architects. Such methods allow an architect to build an abstract picture of an enterprise that makes clear the goals and functionality of the enterprise for different stakeholders.

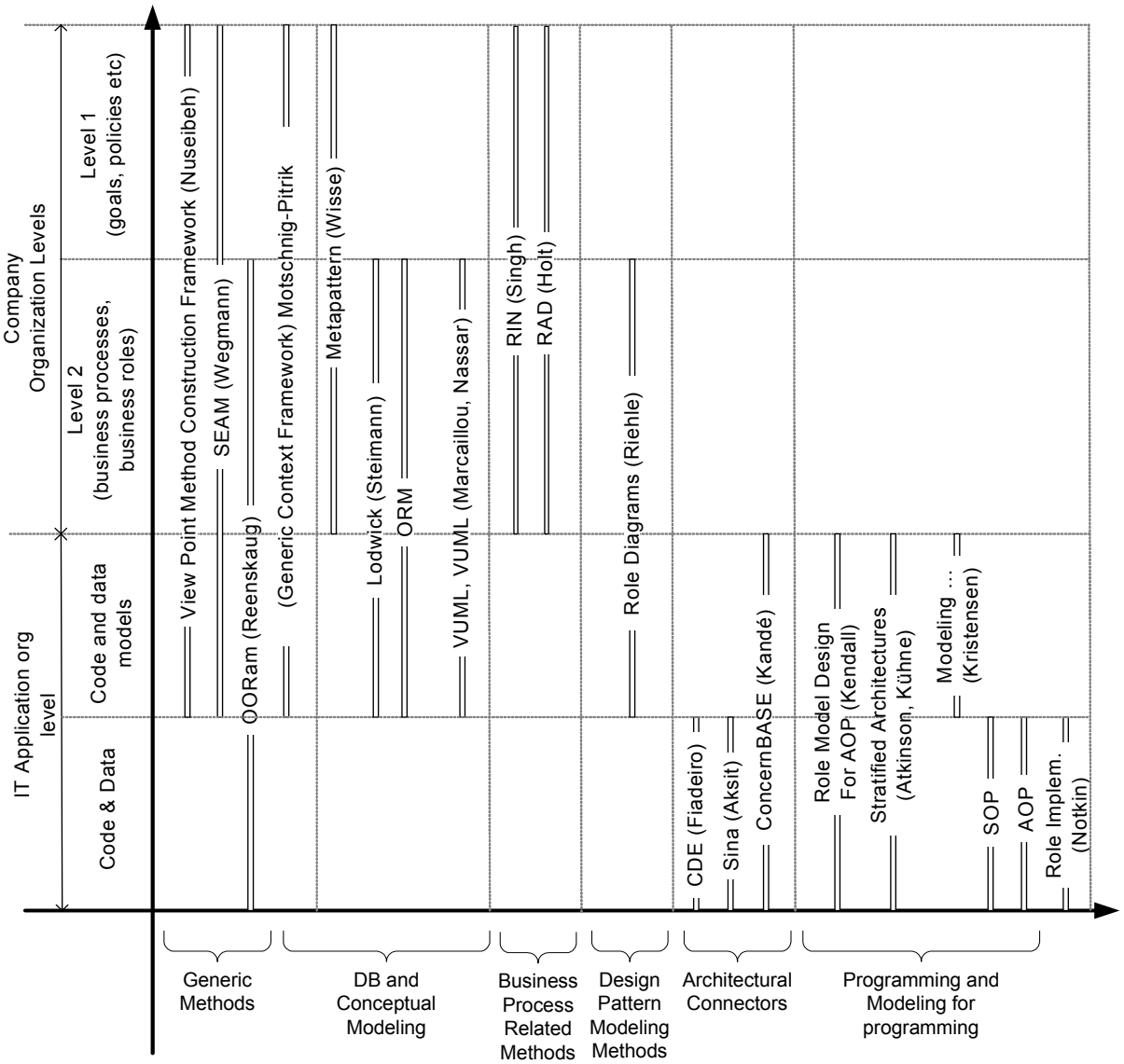


Figure 12. The classification of CBDMs in the context of EA.

2.3 Discussion

The goal of our CBDM classification is to provide researchers, in the first place, and practitioners in the second with the overview of the CBDMs in the field of EA. This overview can be used for many different purposes. However, we want to emphasize two purposes that served as an inspiration for our work:

1. For researchers that use or develop CBDMs, limited only to a particular organization level in EA (like IT application organization level): our classification can help them to position their work in the context of EA and find similar CBDMs based on similar concepts and principles. In addition, this classification can help them to understand the view of different specialist from other EA organization levels.
2. For researchers and practitioners (enterprise architects) that work on generic EA methods (methods that consider all of several organization levels): the goal of such methods is to make the alignment of methods on different levels of enterprise hierarchy. An enterprise architect has to choose appropriate design methods for each organization level and ensure the traceability between these methods. This traceability is possible if the design methods at different organization levels are similar in their basics (their use similar properties and design principles). Our classification can help researchers and practitioners choose appropriate CBDMs such that these CBDMs can be mutually aligned.

The analysis of Table 2 and Figure 12 allows us to make several observations:

1. Only generic CBDMs can be used for modeling at almost all EA organization levels (from marketing to IT implementation). See, for example, OORAM [Reenskaug96] or SEAM [Wegmann03]. However, often these generic methods cannot be used by all specialists in EA team. For example, OORAM or SEAM methods are not very appropriate for programmers who work with concerns because these methods are not capable to represent explicitly all the features of concern-based programming languages (like join points in AOP). Therefore, EA projects are forced to use several CBDMs that have to be aligned. This alignment can be done by means of choosing a set of similar (or compatible) methods for an EA project or by using a generic method as an alignment tool.
2. CBDMs can be divided roughly into three groups:
   - **Conceptual methods**: CBDMs that are used mainly for systems analysis or requirements engineering. These methods deal with conceptual, data, business process and design patterns modeling (see, for example, Metapattern [Wisse01], Lodwick [Steimann00], ORM [Halpin01], Viewpoint UML (VUML) [Nassar03], Role Interaction Nets (RIN) [Singh92], Role Activity Diagrams (RAD) [Holt83], Role Diagrams [Riehle96]).
   - **Technological methods**: CBDMs for code and data design or programming (mostly at IT organization level). In this thesis we have considered CBDMs based on architectural connectors, programming and modeling for programming methods (see, for example, CDE [Fiadeiro97], Sina [Aksit94], ConcerBASE [Kandé00], Aspect-Oriented Design [Kendall99], Stratified Architecture [Atkinson99], Aspect-Oriented Programming (AOP) [Kiczales97], Subject Oriented Programming [Harrison93], Methods for Implementing Roles [VanHilst96]).
   - **Generic methods**: CBDMs that can be used for modeling most of EA organization levels, from marketing to IT implementation (see, for example, View Point Construction Framework [Nuseibeh93], OORAM [Reenskaug96], and Generic Context Framework [Mylopoulos95]).

   These three groups of methods are very different in their objectives. This results in a weak integration between methods from these groups. To increase the integration there is a need in practical EA methodologies that will help researchers and practitioners to understand and integrate objectives of these three groups of methods.
3. Epistemological properties of CBDMs are not developed enough: early CBDMs (earlier than the middle of 90x) were mostly committed in the ontological role of modeling, i.e. these methods aimed to define the main concepts for representing concerns of systems and the semantics of these concepts. Recent CBDMs such as OORAM [Reenskaug96], SEAM [Wegmann03], Metapattern [Wisse01] and others begin to pay attention to the epistemological role of modeling (using organization levels, context, goals, and visual models). In its epistemological role, CBDMs serve as a means to better explain models; they enable building models that can be understood by humans ("human-friendly" models). The problem of making existing approaches more convenient for human reasoning is clearly stated in [Chang99]: "We would like to emphasize informal and yet conceptually precise and practically significant approaches, rather than merely formal languages theory using different formalisms and therefore making them hard to comprehend and compare".
4. Methodological properties of CBDMs are not developed enough: only a few methods support system design verification, consistency checking or model simulation). We understand, however, that the development of methodological properties (especially system design verification) for CBDMs in the context of EA is clearly a difficult task. It requires the integration of different CBDMs (at different organization levels) and the integration of hard (formal techniques) issues with soft (philosophical) issues of modeling.

Part I Summary

In this part of the thesis we have made the analysis of requirements for concern-based design methods (CBDMs) in the context of enterprise architecture (EA). These requirements are synthesized from the principles and properties of the existing CBDMs using the Systems Inquiry. Based on these requirements we have made the classification of twenty most influential CBDMs. This classification can be used by EA researchers and practitioners to choose appropriate methods for their EA methodologies and to compare CBDMs that they use with similar methods.

The analysis of the aforementioned CBDMs shows that many of these methods can be used in the context of EA. However, we see the following two problems with these methods in the context of EA:

- Many of the existing CBDMs support ontological and theoretical principles of concern-based modeling. However, the support of epistemological and methodological principles is often missing. The study of the methodological principles goes out of the scope of this thesis. Therefore, the problem that we are interested in is related to the missing support of the epistemological principles. These principles play an important role in system modeling. They improve the comprehensibility of systems' models. In the

context of EA, the main problem is to understand rather than simply model enterprise processes in order to automate them. The overall understanding of enterprise processes should be based on a method that is not much formal and rigorous but human friendly and convenient for human reasoning. The problem of making existing approaches more convenient for human reasoning is clearly stated in [Chang99]: “We would like to emphasize informal and yet conceptually precise and practically significant approaches, rather than merely formal languages theory using different formalisms and therefore making them hard to comprehend and compare”.

- Although CBDMs look to be very convenient to model systems concerns in a number of specific situations, the definition of concerns is often implicit in CBDMs. Concerns are usually not related to the situations where they are defined. CBDMs do not explain how concerns are identified and how a modeler should advance in a modeling process from the identification of base concerns to the composition of these concerns into the final specification of a system. Without a clear systemic modeling method that guides a modeler, the resulting models are difficult to understand and concerns in these models are difficult to relate one to another.

Note that the requirements for CBDMs that we have considered in this part may not be complete. In spite of this, we believe that the identified requirements and the corresponding analyses of methods are useful in the context of EA from the “pragmatic” point of view: we believe that this part of our thesis will put some order in the mass of concern-based design methods. As a result, we hope that our work will help researches to develop and improve their EA methodologies.

PART II

THE SITUATION-BASED MODELING FRAMEWORK FOR ENTERPRISE ARCHITECTURE

Overview: *Part II is the core of our work where we present our Situation-Based Modeling Framework for Enterprise Architecture. As we have seen in Part I, existing CBDMs do not support the epistemological role of modeling and need a systemic approach for the definition of system concerns. In our framework we solve these problems by means of using the set of epistemological principles that we have identified in Part I. We use these principles in order to define our modeling framework. The structure of this part is the following: In Chapter 3 we give the description of our framework. The definition of our Situation-Based Modeling Framework is accompanied with two examples:*

- *The example of the Simple Banking System (Chapter 4): we concentrate on the illustration of the modeling language and the method of our framework;*
- *The example of the of the Simple Music Management System (Chapter 5): we concentrate on the axiomatic semantics for our modeling language;*

3 Framework Definition

To illustrate our Situation-Based Modeling Framework we use a running example of the Simple Banking System. This example will be used through this chapter. Therefore, we begin with the introduction of this example and with explanations why we choose this example to illustrate our framework.

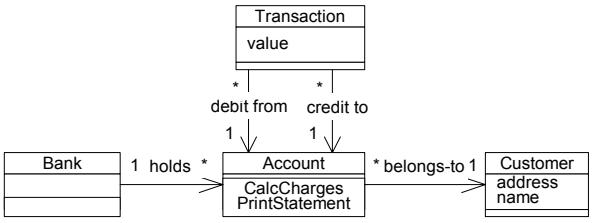


Figure 13. Simple Banking System Object Model

The example of Simple Banking System (Figure 13) is taken from the Ratio Group's white paper [Collins-Cope98] that explains the UML notation for Object-Oriented Analysis and Design. This is a pure UML example that aims to represent main objects that a simple bank has to manage in its work.

We use this example to illustrate the benefits of situation-based modeling framework. Before we begin to introduce our notation, we explain why we are not satisfied with the model from Figure 13. So, what is the problem? The model in Figure 13 was built as a tradeoff between the model's size and its understandability. This model represents some (but not all) concepts that can be used to explain two roles of a Simple Banking System in two different contexts.

First, this model can be used to explain the role of the Simple Banking System in the situation of customer/bank relationship. It can be considered as a model of a bank from the customer's point of view. A customer creates an account in a bank and then makes credit/debit transactions with its account. Note that in order to explain the relation between a customer and a bank, a customer does not need to know that a bank holds many accounts. Therefore only one account can be modeled.

Second, this model can be used to explain a role of the Simple Bank System in the situation where there are multiple accounts. However, in this model, all these bank accounts are not related to each other explicitly. Each account is responsible only for keeping its own balance. We cannot see the whole idea of a bank: how a bank accumulates money from depositors and invests it.

We can see that the model in Figure 13 is a mixture of concepts used to reason about different roles of the Simple Banking System. Concepts are not specified with roles and corresponding situations they belong to, which complicates the overall comprehension of a model.

We use our framework to propose an alternative model of the Simple Banking System that is conceptually more rigorous and makes reasoning about the Simple Banking System easier.

Let's continue with the description of the structure of this chapter where we define our Situation-Based Modeling Framework based on the first three[12] directions of the Systems Inquiry (see Introduction): the philosophy, the theory and the methodology:

- In Section 3.1 we explain the philosophy where we define main modeling concepts and a modeling language for our modeling framework.
- In Section 3.2 we explain the theory where we show how a system is modeled as the composition of system roles that are related to different situations.
- In Section 3.3 we define the methodology of our modeling framework: i.e. a modeling method as a set of constraints that a modeler has to respect when specifying systems with our method. As a part or our method we also define the axiomatic semantics for our modeling language.

3.1 Framework Philosophy

The framework philosophy consists of two parts: the ontology and the epistemology of our modeling framework.

The ontology studies the concepts that we need to represent reality: here we define the main concepts for our modeling framework. In order to give rigorous definitions for these concepts, we use the ISO/ITU standard

[12]The fours direction, the application of our framework, is given in Part III of this thesis.

"Reference Model for Open Distributed Processing" – part 2 [ISO96]. To make the definitions from this section less ambiguous we map these definitions to the set theory.

Based on RM-ODP, modeling consists of identifying entities in the universe of discourse and representing them in a model. The *universe of discourse* (UoD) corresponds to what is perceived as being reality by a business analyst and *entity* is "any concrete or abstract thing of interest" [ISO96] in the UoD. Identified entities are modeled as *model elements* in a *model*. Model elements are different modeling concepts (object, action, behavior etc).

We continue this section with the definitions of concepts that we need to describe the information viewpoint of a computational object. We begin with the definition of a computational object:

Object: "An object is characterized by its behavior and dually by its state" [ISO96].

The duality of state and behavior means that the state of an object determines the subsequent behavior of this object. The definition of an object is based on the definition of behavior and state:

Behavior: A collection of actions and a set of (sequential) relations between actions.

State: A collection of attributes, attribute values and relations between attributes.

Attributes can change their values; relations between attributes can be created or deleted. To specify these changes we use pre- and postconditions. We will also need the concept of time to specify the ordering of actions. For any action we will specify "*an interval of arbitrary size in time at which action can occur"* [ISO96]. We group time intervals into the partially ordered set of time points. This set of time points defines sequential relations between actions. Thus the behavior of an object we describe as a tuple:

$\vartheta = <A, SeqRels, T, AVals, Attrs, AttrRels, Pre, Post, instant\_begin, instant\_end, precondition, postcondition>$

where:

$A = \{a_1,a_2,...\}$	is a set of actions
$SeqRels: A,A \rightarrow \{true, false\}$	is a total function[13] that defines sequential relations between actions
$T = \{t_1,t_2,...\}$	is a partially ordered set of time points
$AVals=\{val_1,val_2,...\}$	is a set of attribute values
$Attrs=\{attr_1,attr_2,...\}$, where $attr_n: T \rightarrow AVals$	is a set of attributes, where each attribute is a total function that returns the attribute value for a given time
$AttrRels=\{rel_1,rel_2,...\}$, where $rel_n: T \rightarrow Attrs, Attrs$	is a set of relations between attributes, where each relation is a function that returns two attributes for a given time
$Pre=\{pre_1,pre_2,...\}$	is a set of pre-conditions (predicates defined on the set of attributes and set of values)
$Post=\{post_1,post_2,...\}$	is a set of post-conditions (predicates defined on the set of attributes and set of values)
$precondition: A \rightarrow \{Pre\}$	is a total function that returns the set of precondition predicates for a given action
$postcondition: A \rightarrow \{Post\}$	is a total function that returns the set of postcondition predicates for a given action
$instant\_begin: A \rightarrow T$	is a total function that returns the instant in time when the action starts
$instant\_end: A \rightarrow T$	is a total function that returns the instant in time when the action ends

Let's give some comments about pre- and postconditions. They are defined as predicates on the set of attributes and their values. For example:

$predicate_1(attr_n, time_k, val_m)$: $attr_n(time_k)$ *"is equal to"* val_m ;

This predicate allows us to define the following precondition for some action *a*:
$pre=predicate_1$(name, a.instant_begin, « pavel »)
This precondition means that the *a* action can be performed if the *name* attribute is equal to "*pavel*".

The ontology of our framework is influenced by its epistemology. The epistemology studies the principles that explain how models are understood by humans. We need to consider epistemological principles in order to define the main concepts of our framework. Let's see them. These principles are composed from the list of epistemological principles that we have seen between the existing CBDMs (see the state of the art section in Chapter 1). In Table 3 we show these principles. In this table we also explain why certain principles from Chapter 1 are not included as the basis for the modeling language of our framework.

[13] Total function is a function that is defined for each value of the input set.

Table 3. Epistemological Principles and their Application for our Modeling Language

#	Name of concepts, properties or principles	Description	Will be used or not for the definition of our modeling language? Why will not be used?
R 1.2	Organization levels	Views of different specialist that collaborate in an EA project.	Will not be used: not relevant because in our framework we model systems only at a given organization level
	Scoping views	Used to extract (or filter) certain information from an existing design.	Will not be used: scoping views can be applied in future as an extension of our framework. In particular, scoping views can be useful for the development of the tool that support modeling with our framework.
R 1.3	Situation relativity principle	CBDMs should support explicit modeling of situations.	Will be used: see in the text of this section.
R 1.4	"Goal driven" principle	A goal represents the results to be achieved by a system, placed in a given situation.	Will be used: see in the text of this section.
R 1.5	State-Behavior Holism Principle	CBDMs should not tend to separate the state and behavior information of a described system	Will be used: see in the text of this section.
R 1.6	Principle of Diagrammatic Representation	Models build in the context of EA should be simple and diagrammatic.	Will be used: see in the text of this section.

The principle R1.3 states that CBDMs should support the explicit modeling of situations. This principle reflects a fact that systems are modeled differently depending on the situations in which they are considered: a model element stands for an entity from the UoD, not in all respects, but in reference to a sort of idea or situation. Therefore, we consider situation as a first class citizen of our framework.

In order to make situations the first-class citizens in our modeling framework, we took the key principles for our framework from the situation theory. The situation theory [Barwise83] is a mathematical theory of meaning which clarifies and resolves some problems in the study of language, information, logic, philosophy, and the mind [Barwise83]. Below we give a shot introduction to the situational theory where we underline principles that we have used in our work.

Situational theory overview

"Situation Theory is a powerful addition to logic, developed by logicians, mathematicians and linguists at Stanford's Center for the Study of Language and Information. At some cost, it addresses the ability to represent "situations" as first order citizens in logic".

[Cited from http://alfve.sourceforge.net/glossary.html on Jan. 21, 2005]

In Situation Theory every assertion occurs in a situation. Situations are described as a set of infons. "*Infons* [correspond to collaborations in our terminology] are the basic unity of information. The simple (or basic) infons convey information about individuals related in some way.

The structure of an infon consists of a relation, the related individuals [correspond to computational objects in our terminology], and the roles each one plays in the relationship. The individuals of an infon are called arguments of the infon" [Pinheiro02].

If a situation S makes a certain assertion ass true, this is denoted in Situation Theory as: $S \models ass$

To give an illustration of Situation Theory we give examples (taken from [Pinheiro02]) in the PROSIT language that implements the theory:

```
(insert_into client card ATM)   // represent the insert_into infon with three objects: client, card, ATM
insert_into (client, card, ATM) // the same infon in a more readable form
```

A situation is described using every infon that has to be held in that situation.

In Situation Theory situations are usually structured in hierarchical way. For example, in PROSIT there is one universal situation called *top* that holds all infons. To create a new situation S, the (in S) command should be used. For example:

```
(In withdraw)                    // in the current situation create a new situation called withdraw
(! (insert_into client card ATM) )// that extends the current situation with the insert_into infon
```

This command makes withdraw the current situation. Then the lower level situations can be created in the withdraw situation in the same way.

"In Situation Theory constraints, constraints model relationships which are defined between types [such as card or ATM]" [Huibers96]. If situation are not independent (i.e. they overlap in some way) than there should be some kind of information flow between these situations. The nature of this flow is defined by the constraint: $\phi \rightarrow \varphi$, where ϕ and φ are types from the overlapped situations.

The direct application of the Situation Theory is not feasible in the context of EA because this theory is quite formal and too much adapted to linguistic needs. However, we took some basic principles (they are underlined in the situation theory introduction insertion above) from this theory as the foundations of our framework:

1. Situation-based modeling: Any entity (any system) in the UoD should be considered in a number of situations. Any assertion about system properties depends on a situation.
2. Hierarchical modeling situations: situations are structured in a hierarchical way. Correspondingly, the behavior of an enterprise system should be represented in the form of the role hierarchy that corresponds to the hierarchy of situations.
3. Explicit modeling of constraints: the mutual relations between system roles should be modeled explicitly.

Throughout this chapter we show how these principles are realized in out framework. In this section we begin with the first principle. The second principle is explained in Section 2, where we show how different roles are composed together. The third principle we explain in Section 3, where we explain a modeling method of our framework.

The first principle in our framework is realized in the following way: we model a system as a set of roles. Each role represents the behavior of a system related to a given situation. Any assertion about a role (such assertions about state and behavior of a role) is valid only in this given situation.

The notion of role that is central for our Situation-Based Modeling Framework. Let's note that the meaning of the term "role" can be understood differently depending on the background of our readers. In Appendix 2 we show that the term "role" can be understood as a behavior or an object. In our framework role is understood as a

behavior of a computational object: it defines the partial behavior of an object related to a certain situation. We define the term role[14] in the following way:

Role: "An abstraction of the behavior of an object" intended for achieving a certain common goal in collaboration with other roles.

We understand collaboration as a joint action from the Catalysis method [D'Souza98].

Collaboration (or a joint action) is "a change in the state of some number of participant objects without stating how it happens and without yet attributing the responsibility for any of it to any one of the participants" [D'Souza98].

We model a role with a tuple R_n (similar to ϑ):

$R_n = <A_n, SeqRels_n, T_n, AVals_n, Attrs_n, AttrRels_n, Pre_n, Post_n, instant\_begin_n, instant\_end_n, precondition_n, postcondition_n >$[15]

where sets of actions, constraints, time points, etc are related to a given situation. This gives us:

$R_n = abstraction(situation_n, ϑ)$

In our work we use the definition of situation inspired by the situation theory and the OOram's "role model" definition [Reenskaug96]. This definition allows us to make explicit external roles that can influence the behavior of a considered object:

Situation is the set of collaborating roles ($\{R_1, R_2, ..., R_n\}$) along with their state and behavior.

I.e. a situation can be seen as one or several collaborations between these roles.

The last definition that we need to define a role (see above) is:

Goal (of a system in a certain situation) is a set of all postconditions for the corresponding role.

$G_n = \{Post_n\}$ *for role* R_n

Let's come back to the set of epistemological principles. Using epistemological principles as a basis for our modeling framework makes the understanding of our models easier. We have already listed the epistemological principles that we use in our framework (see Table 3). Now we explain how they are used.

We begin with the "Principle of Diagrammatic Representation" that asserts that our modeling language should be visual. Therefore, we continue this chapter with the definition of our visual language that we use in our Situation-Based Modeling Framework.

We explain the nation of our modeling language with the models of very basic roles that correspond to basic situations in which the Simple Banking System (see the description in the beginning of this chapter) can be found. These roles will be composed into bigger roles throughout this section. In our notation, we represent a situation (a set of collaborating roles) by a rectangle that includes a set of collaborations (dashed ovals), set of roles (stick men) and role names (names below stick men). The name of a situation is given in the upper part of a box. We represent objects with cubes; object names are given below cubes.

[14] This definition is inspired by the definition of Role in RM-ODP [ISO96] and in [Genilloud00].

[15] To reference elements in the tuple R_n we will use the following notation: $Set_n.Element_y$, where Set_n can be any set from the tuple R_n. For example, $R_1.precondition_1(A_1.a_1)$ is a $precondition_1$ for the a_1 actions in the specification of the role R_1.

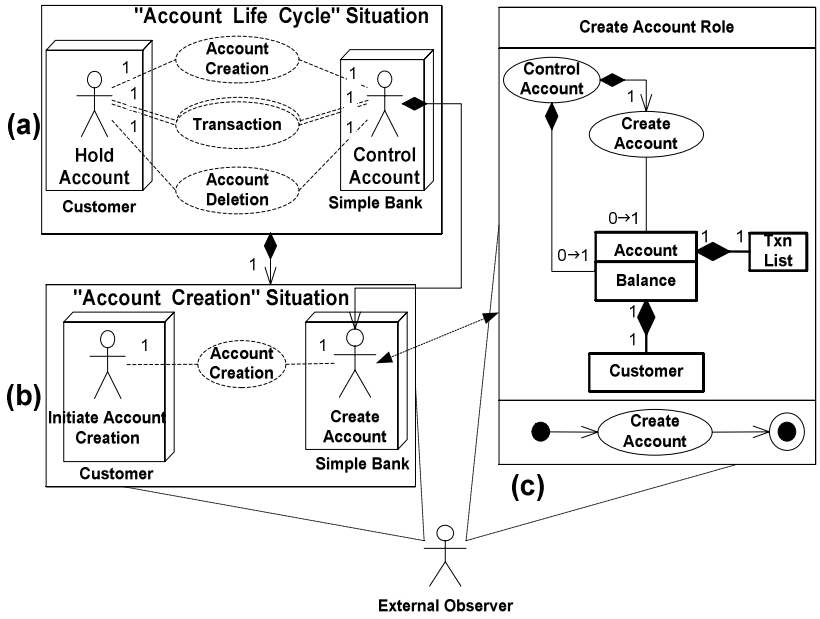


Figure 14. Roles of a Simple Bank object in different situations (a) "Account Life Cycle" situation; (b) "Account Creation" situation; (c) Detailed specification of the "Create Account" role

Each role in a collaboration can be specified in details (see an example in Figure 14.c). A detailed specification consists of a box with three panes. This notation is similar to the representation of a class in UML. The difference is that instead of an attribute compartment in UML (middle pane in each box) we use graphical notation based on a UML class diagram. It contains not only attributes and relations between them, but also actions. This idea of putting actions and attributes in the same section matches the R 1.5 principles of our framework (see Table 3). This is the *State and Behavior Holism principle*: the state (or data) and behavior models should be integrated in the context of EA. It allows us to fulfill the following goal:

- We show the life cycle of the information objects: i.e. how information (modeled as attributes and relations between them) is created, persisted and deleted in our models. For this purpose we show explicitly actions that are responsible for creation, persistence and destruction of attributes.

In the following table we explain the semantics of relations between actions and attributes[16]:

Table 4. Semantics of relation between actions and attributed in the SEAM Visual Language

super-action — m — sub-action	The super-action includes m sub-actions. This means that m instances of an action happen in the life cycle of super-action.
action — 0→1 — attribute	Action is responsible for the creation of an attribute (attribute multiplicity changes from 0 to 1). We emphasize newly created attributes and relations with thick lines (postconditions).
action — 1→0 — attribute	Action is responsible for the destruction of an attribute (attribute multiplicity changes from 1 to 0).
action — (1→1) — attribute	An attribute is updated by an action. Showing the attribute multiplicity (1→1) is optional.
action — attribute	An attribute is persistent in an action

Each role in the middle pane should contain at least one action of the same name (such as the "Create Account" action in the "Create Account" role from Figure 14.c).

In the lower pane of each box, in the compartment that holds a list of operations in UML, we represent the behavior of a role based on the notation for UML activity diagrams.

Note that in our modeling framework for any role of a system the corresponding situation should be defined. This is done in order to make modeling assumptions about situations explicit (this reflects the situation relativity principle, see Table 3). For example, for the "Create Account" role, the "Account Creation" situation should be defined.

[16] Note that our notation can be confusing for people having experience with Entity Relationship Diagrams (ERD). Attributes in our visual language are very similar with entities from ERD. However, in our models ovals represent actions but not attributes of entities as in ERDs.

Figure 14.b gives an example of a model of a Simple Bank object in the "Account Creation" situation where a customer creates an account in a simple bank. With the whole-part relationship between situations (between Figures 14.a and 14.b) we show that the "Account Creation" situation is a part of the "Account Life Cycle" situation. This means that the "Account Life Cycle" situation includes all collaborations and objects from the "Account Creation" situation and that the "Control Account" role includes the "Create Account" role.

The "Account Creation" situation is modeled with two roles: "Initiate Account Creation" and "Create Account" and limits the scope of the model to communications that can be observed by an External Observer between the Customer and Simple Bank objects. These communications have a final goal of creating an account by the Simple Bank object. Based on the *"Goal driven" principle* (see Table 3) we want to show this goal explicitly in out visual language. This goal is reflected in a diagram from Figure 14.c. An Account is created with TxnList (transaction list) and Customer attributes. We show newly created attributes and relations with highlighted, thick lines. Note that creating an account is not an end in itself. This account will be further used by the Customer object in the "Account Life Cycle" situation (Figure 14.a) for making debit and credit transactions and then for the account deletion. To reflect a fact that a created account will be used in a "higher"- level situation, we include the "Control Account" action in the middle pane of the "Create Account" role (Figure 14.c). We connect this action to the Account attribute with a thick line (to emphasize postconditions). This line shows that Account, Customer and TxnList attributes will be used in another situation.

3.2 Framework Theory: Composition and Composition constraints

This section describes the third direction of the system inquiry: the theory of our framework. The systems theory defines the most general theoretical properties of our framework. As we already explained, our framework describes systems as a set of roles. These roles represent the abstraction of system behavior related to different situations. We believe that there are two most common forms of abstraction[17]: generalization and composition. Therefore, we consider generalization and composition as two the most common properties of all CBDMs. Generalization is a powerful tool that have been studied by many different researchers (see Chapter 1 of Part I). Currently it is widely used in system modeling. In our work we concentrate on a composition. Therefore, the theory of our framework studies how a system is modeled as the composition of system concerns (we remind you that we call them *roles* in our framework) that are related to different situations.

In this section we will refine the second principle taken from the situation theory. This principle is related to the composition of roles: the mutual relations between system roles [in a composition] should be modeled explicitly.

3.2.1 Identity and Composition Constraints Semantics

In our framework, the overall behavior of a system is defined as the composition of roles. If these roles are independent, than the behavior of this system is simply the bug of roles. However, roles are rarely independent because situations, where these roles are defined, usually intersect. What does a mutual dependence between roles mean? To explain the meaning of the mutual dependence of roles we use Figure 15. The upper part of Figure 15 shows the Universe of Discourse, the lower part represents the model of the universe of discourse. Let's suppose that we have a business analyst who modeled separately two roles of the Simple Banking System: "Create Account" and "Make Transaction" (defined correspondingly in the "Account Creation" and "Transaction" situations, see Figure 15). We can see that the business analyst decided to model Entity1 as the "Create Account" role (where he modeled Entity5 as the "Create Account" action and Entity6 as the "Account" attribute). He also decided to model Entity2 as the "Make Transaction" role (where he modeled Entity3 and Entity6 as attributes and Entity4 as the "Make Transaction" action). To make a bigger model he wants to compose these two role models. If we consider the semantics of role composition we can see that roles are mutually dependant, i.e. behavior of different roles can influence behavior of other roles involved in composition. By looking in the Universe of Discourse we can see that the "Account" attribute of the "Create Account" role and the "Account" attribute of the "Make Transaction" role have the same semantic meaning since they model the same Entity6. Also we can see that the "Make Transaction" action (Entity4 in the UoD) follows the "Create Account" action (Entity5 in the UoD). In our framework we claim that these kinds of dependencies between roles should be modeled explicitly in the composition of roles.

[17] Friedrich Steimann in [Steimann03] considers three main forms of abstraction: composition, classification and generalization. However, in our work we consider classification as the result of generalization: instances become classified based on defined types and subtypes.

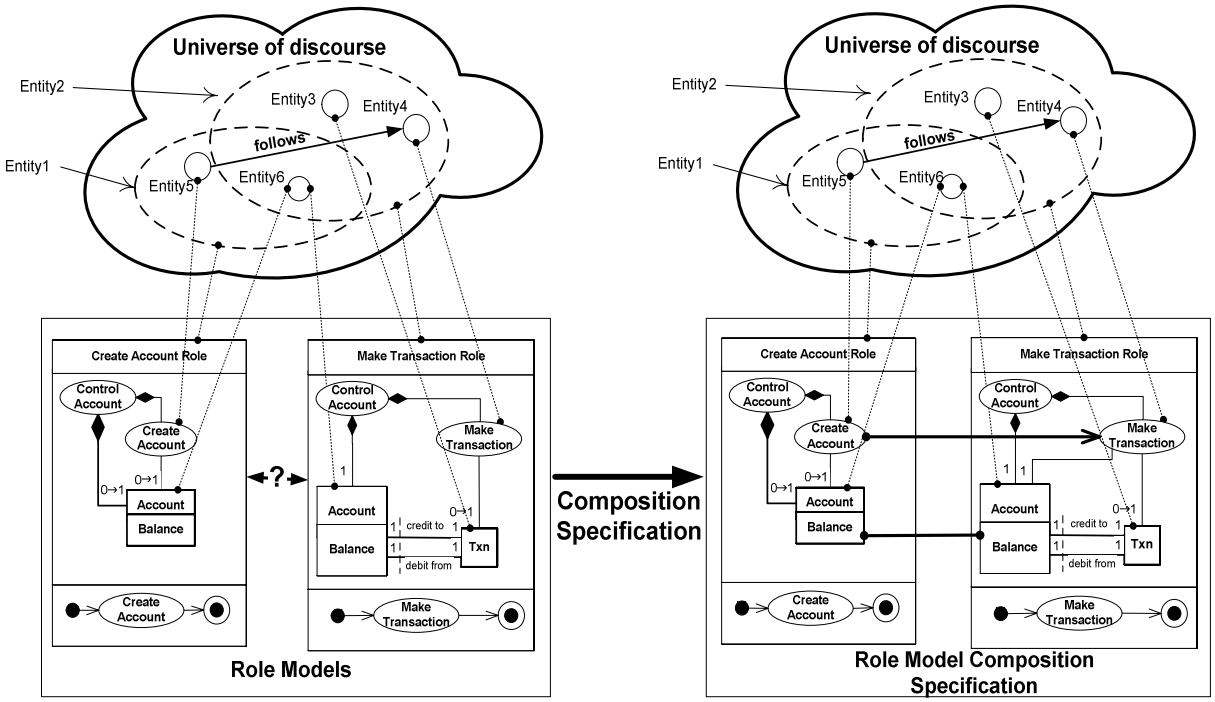


Figure 15. The semantics of composition constraints; a: Two base role models and their semantic meaning; b: Role Model Composition Specification: base role models and composition constraints.

The dependence between roles can be specified in several ways. Examples of different syntax for expressing role's mutual influence can be seen in [Riehle98], [Fiadeiro01], [Reenskaug96]. But the syntax proposed in these works depends on implementation details: composition is specified as a communication through given role's interfaces. These technical details complicate the understanding of composition semantics. That is why we believe that the syntax abstract from the implementation details can be useful in the modeling of role composition. This can help a modeler to concentrate his efforts on design decisions. Decisions taken by a modeler have to be based on the semantics of the composition rather than on implementation. In the next subsection we define a *Role Model Composition Specification* that reflects the semantics of composition and makes abstraction from the implementation details.

3.2.2 Role Model Composition Specification

Our Role Models Composition Specification is based on the concept of *composition constraints*. As we have seen in the previous sub-section, we have to define a new concept that will specify how base roles influence each other within a new composed model. Composition constraints are used for this purpose:

Composition constraints: *Constraints implied on the behavior of base roles*[18] *defined for different situations.*

In general, composition constraints can be specified as a mapping of model elements used in the specifications of the basic roles.

Let's R_1 and R_2 be two base roles. In this case composition constraints can be defined as a set of couples that show how mapping is done:

$Composition\ constraints = \{c_1, c_2, ...\}$, where

$$c_n \overset{def}{=} (R_1.Set_x.element_n \leftrightarrow R_1.Set_x.element_m) \quad or$$

$$c_n \overset{def}{=} (R_1.function_y(R_1.Set_x.element_n) \leftrightarrow R_1.function_y(R_1.Set_x.element_m))$$

where Set_x can be one of the following sets: *A, T, AVals, Attrs, AttrRels, Pre or Post* and $function_y$ can be one of the following functions: *instant_begin, instant_end, precondition, postcondition.*

Note that the specification of composition constraints between certain modeling elements may require the specification of additional constraints. For example, if we want to specify the identity of two actions from two different roles, we also have to specify that the preconditions and the postconditions for these actions are the

[18] We call role models before composition *base role models* and correspondingly roles, *base roles*. We call role models after composition *composed role model* and correspondingly roles, *composed roles*.

same. In the general case, this rule can be formulated in the following way: if two elements ($R_1.Set_x.e_n$ *and* $R_2.Set_x.e_m$) from base roles (R_1 and R_2) are mapped to each other (e.g. role1.A.action1 is the same as role2.A.action2) and there is a function (e.g. precondition) in the definition of base roles using these elements as arguments (e.g. precondition(action1)), then the resulting elements of these functions should also be included in the composition constraints (e.g. role1.precondition(action1) same as role2.precondition(action2)). This rule can be described as:

$$(c_n = (R_1.Set_x.e_n \leftrightarrow R_2.Set_x.e_m)) \vee (R_1.f(R_1.Set_x.e_n) = R_1.Set_y.e_p) \vee (R_2.f(R_2.Set_x.e_m) = R_2.Set_y.e_q) \Rightarrow c_{n+1} = (R_1.Set_y.e_p \leftrightarrow R_2.Set_y.e_q).$$

Below we give definitions of four different composition constraints that we have used in our research work[19]. These composition constraints respect the consistency rule shown above:

- **Attribute Identity** ($Role_1.Attrs.attribute1 \bullet\!\!-\!\!\bullet Role_1.Attrs.attribute2$)

  $\forall t1 \in Role_1.T, \forall t2 \in Role_2.T:$

  $(Role_1.Attrs.attribute1(t1) \leftrightarrow Role_2.Attrs.attribute2(t2))$

  This means that values of attribute1 and attribute2 should be equal at any time moments specified by Role1 and Role2 correspondingly.
- **Synchronized Actions** ($Role_1.A.action1 \rightarrow\!\bullet\!\leftarrow Role_2.A.action2$).

  $Role_1.instant\_begin(Role_1.A.action1) \leftrightarrow Role_2.instant\_begin(Role_2.A.action2)$

  $Role_1.instant\_end(Role_1.A.action1) \leftrightarrow Role_2.instant\_end(Role_2.A.action2)$

  This means that the actions of each role start and finish at the same time.
- **Constraints of Sequentiality** ($Role_1.A.action1 \rightarrow Role_2.A.action2$).

  $Role_2.instant\_begin(Role_2.A.action2) \leftrightarrow t):$

  $t \in followingTE(Role_1.instant\_end(Role_1.A.action1))$

  This means that the second action can start at any time after the completion of the first action. In details about constraints of sequentiality you can read in [Balabko01].
- **Action Identity** ($Role_1.A.action1 \bullet\!\!-\!\!\bullet Role_2.A.action2$).

  $Role_1.A.action1 \leftrightarrow Role_2.A.action2$

  $Role_1.instant\_begin(Role_1.A.action1) \leftrightarrow Role_2.instant\_begin(Role_1.A.action2)$

  $Role_1.instant\_end(Role_1.A.action1) \leftrightarrow Role_2.instant\_end(Role_2.A.action2)$

  $Role_1.precondition(Role_1.A.action1) \leftrightarrow Role_2.precondition(Role_2.A.action2)$

  $Role_1.postcondition(Role_1.A.action1) \leftrightarrow Role_2.postcondition(Role_2.A.action2)$

  These composition constraints specify two actions that have the same results (and thus can be considered identical), i.e. that both roles specify actions with the same pre- and post-conditions and happening at the same time. For example, “Identical actions” can specify that both roles move the same entity in the universe of discourse from one place to another and these moves happen in the same time interval.

3.2.3 Multi-Situational View

Role model composition specification (base roles plus composition constraints) helps a modeler to reason about attributes and actions from roles that are related to different base situations. However, the role model composition specification does not help a modeler to reason about all base role as a whole, without looking into details of each base role. In order to do this, we define a *multi-situational* view.

A multi-situational view shows the result of the composition of base roles. The multi-situational view includes (see an example in Figure 16):

- Set of base roles represented as actions (ovals in the middle and lower panes of the role model)
- Set of relations between attributes and base roles (shown with dashed lined arrows). This helps the reader of a diagram to understand the meaning of the attributes for the base roles. If the meaning is not clear, a reader can always refer to a detailed model of the base roles.
- Role hierarchy (the same as we discussed earlier).
- Multiplicities for roles and attributes (the same as we discussed earlier).

[19] This list is not exhaustive and can be extended with other useful composition constraints.

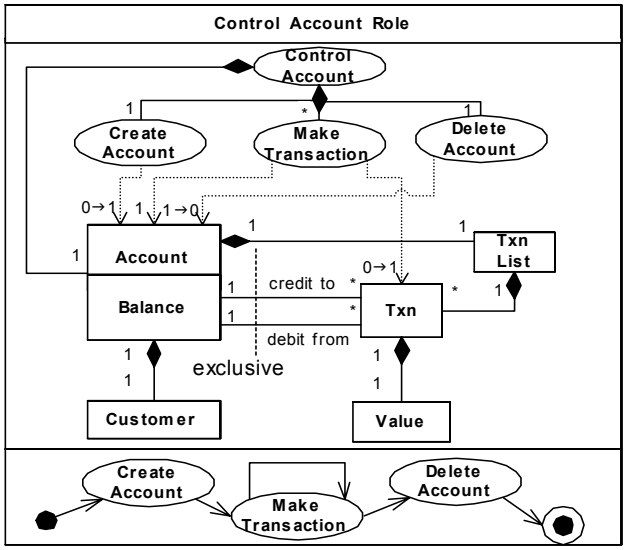


Figure 16. Multi-situational view: the "Control Account" role

In Figure 16, for example, we can see the multi-situation view for the "Control Account" role. It specifies that the "Control Account" role has its three sub-roles. For each sub-role we can see how it influences the attributes of a system. An attribute can be: created (multiplicity changes from 0 to 1), used in some way (multiplicity equal to 1) or deleted (multiplicity changes from 1 to 0). Therefore the life cycle of each attribute becomes explicit. The explicit life cycle makes reasoning about attributes and base roles in our visual models easier.

Note that our notation is based on UML. This allows us to borrow from UML some useful diagram elements such as the exclusive (or disjoint) constraint that we used in Figure 16 to show that any transaction can be placed only in one set: Account.credit_to or Account.debit_to.

3.3 Framework Methodology: Method and Model Checking

In this section we present a method that explains how models are built and checked in our modeling framework. We show how the three principles taken from the situation theory (see Section 1.1.1) are applied in our framework. We continue this section in the following way: In Section 3.3.1 we give an overview of our method, in Section 3.3.2 we explain the axiomatic semantics that is used to check the correctness of our models.

3.3.1 Overview of a Method

In our method we do not prescribe how a modeler should build models. We only give modeling constraints that a modeler should satisfy when using our method:

1st constraint: A system of interest is considered in a number of meaningful situations. Each situation should be modeled as a group of collaborating objects with a certain goal. We recommend using the "Create Collaboration, Do Collaboration, Delete Collaboration" pattern. This pattern makes explicit the life cycle of collaborations between roles. Using this pattern we can specify how a collaboration is created, how it is used and how it is deleted. For example, in Figure 14 we have specified the "Account Life Cycle" situation with three types of collaborations: "Account Creation", multiple "Transactions" and "Account Deletion".

2nd constraint: The hierarchy of situations should be specified using the whole-part relation. It specifies the situation containment and the multiplicities of the situations in the situation containment. For example, in the model of the Simple Banking System we have to specify that the "Account Life Cycle" situation contains one "Account Creation" situation, multiple "Transaction" situations and one "Account Deletion" situation.

3rd constraint: For each situation (a group of collaboration roles) a separate model of a system of interest should be built. For example, in the "Account Creation" situation a model of a Simple Bank object is given in Figure 14.c.

4th constraint: Based on the hierarchy of situations, a composition of roles that correspond to the lower level situations should be specified. For example, the "Control Account" role (in the "Account Life Cycle" situation)in the model of a Simple Bank System is composed of one "Create Account" role, multiple "Make Transaction" roles and one "Delete Account" role. Roles are composed basically by finding identical attributes and putting compositional constraints.

Note that these constraints almost one-to-one reflect the principles that we have taken from the situation theory (see Section 1.1.1). The first constraint of our modeling method reflects the first ("situation-based modeling") principle. The second and third constraints reflect the second ("hierarchical modeling") principle. The third constraint reflects the third ("Explicit modeling of constraints") principle.

As we have already explained above, the whole idea of our method is to specify base roles that are small enough to reason about them and then compose them into bigger roles. This way of modeling can be considered as modeling by means of "examples"[20]. I.e. base roles can be considered as examples of system behavior in different situations. Ones these roles-"examples" are specified, the specification of the whole system is defined as the composition of them. This composition is mainly based on the discovery of identical modeling elements and composition constraints between modeling elements in roles-"examples" that have to be composed.

3.3.2 Model Checking Based on Axiomatic Semantics

The design of systems should be based on adequate models. Only adequate models can ensure that a system evolves in correspondence with the needs of system's users. The understanding of model adequacy requires the interaction between system developers and customers (or other system stakeholders). They are not experts in system modeling and may have problems in understanding semantics of specification diagrams (such as UML class or statechart diagrams). These diagrams are convenient for professionals and represent the generalizations of many examples (or scenarios) from the problem domain. On the contrary, for customers it is more convenient to talk in terms of particular examples (or instances) of models.

In this Section we propose using axiomatic semantics for the generation of model instances. Axiomatic semantics allows a modeler to check the correctness of a model (or a program) by means of defining the set of invariants on the state of a system. These invariants are usually formalized with the predicate logic.

We use the Alloy modeling language[21] as formalism for the specification of the axiomatic semantics for our modeling language. The Alloy language is very convenient to specify and analyze the static invariants on the sate of a system. However, the analysis of the dynamic invariants (system changes or actions) is not very convenient in Alloy. Other formalisms that reflect operational semantics such as Abstract Sate Machine (ASM) should be used for this purpose. This is a separate research topic covered by other researchers in our laboratory (see [Rychkova03]).

Let's return to Alloy. The mapping of our modeling language to the Alloy code is quite straightforward:

- Attributes are mapped to the Alloy types (or sets)
- Relations between attributes are mapped to the relations between the Alloy types (or sets)
- Multiplicities are mapped to the Alloy facts (formulas that constraint the values of the sets)
- Additional invariants or constraints are mapped to Alloy facts (formulas that constraint the values of the sets)

Let's look at the example in Table 5. It shows a state structure of the Control Account role model (Figure A in the table). This state structure is mapped to the Alloy code (see Code Fragment in the table). This Alloy Code was used to generate the instance diagram (Figure B in the table) that can be used for the analysis of the Control Account role model. The instance diagram was generated using the Alloy Constraint Analyzer. It checks the consistency of a formal Alloy model; randomly generates a solution (an instance of a model) and visualizes it. As we can see, the automatically generated instance diagram (Figure B in the table) adequately represents the reality. It represents an Account for one customer with two transactions (creditTxn and debitTxn). Other instance diagrams can also be generated and checked.

[20] An important aspect in the context of modeling by means of "examples" is to make the abstraction of certain details that may be out of the scope of modeling. For example, let's suppose that a modeler specified two actions *a* and *b*. Let's suppose that these two actions are similar and the only difference between them is the time interval when these actions occur. In certain cases a modeler may be interested in making abstraction of the actual time of action occurrences. In this case a "time abstracted type" can be specified that represents both these actions. In Appendix 3 we have considered how to define different kinds of such "time-abstracted" modeling elements (such as time-abstracted preconditions, postconditions, actions and etc.).

[21] "Alloy is a language for describing structural properties. It offers declaration syntax compatible with graphical object models, and a set-based formula syntax powerful enough to express complex constraints" [Jackson00]. See also http://sdg.lcs.mit.edu/alloy/.

Figure A

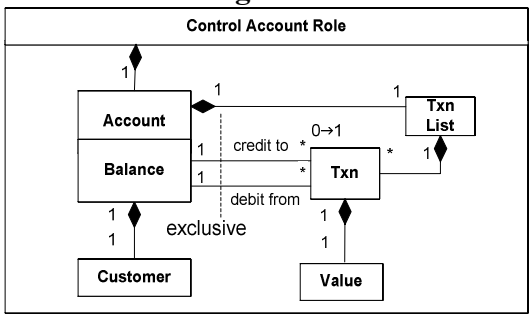
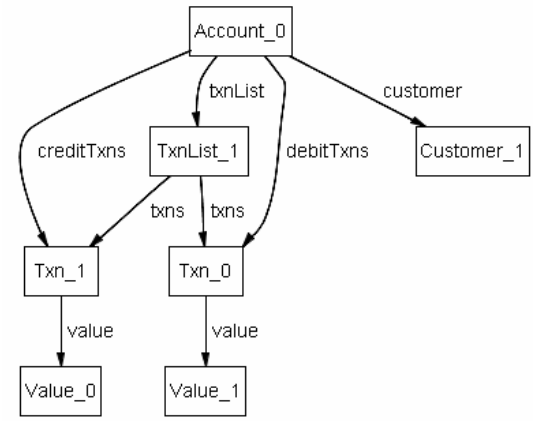


Figure B



Code Fragment

```
1.  module ConrtolAccountRole   //module declaration

2. sig Account { txnList: TxnList,  customer: Customer,
                 creditTxns: set Txn,  debitTxns: set Txn }
3. sig Customer {}
4. sig TxnList {txns: set Txn}
5. sig Txn {value: Value}
6. sig Value {}

//---------- Multiplicity Facts ----------
7.  fact {   // TxnList (1)->(*) Txn
      all t:Txn | one tl:TxnList | t in tl.txns}

8.  fact {   // Account (1)->credit(*) Txn or (exclisive)
             // Account (1)->debit (*) Txn
      all t:Txn | one a:Account |
                  ((t in a.creditTxns) or (t in a.debitTxns)) &&
                  not ((t in a.creditTxns) && (t in a.debitTxns)) }

9.  fact {     // Txn (1)-> (1) Value
      all v:Value | one t:Txn | v in t.value }

10. fact {     // Account (1)-> (1) Customer
      all c:Customer | one a:Account | c in a.customer }

12. fact {     // Account (1)-> (1) TxnList
      all tl:TxnList | one a:Account | tl in a.txnList }

//---------- End of Multiplicity Facts ----------
```

Table 5 Alloy-based axiomatic semantics for the Control Account role
a: Diagram that represents state structure of the Control Account role
b: Corresponding Alloy code
c: An instance diagram automatically generated with the Alloy Code Analyzer

Let's see how the mapping between the model of the Control Account role and the Alloy code is done: all rectangles (attributes) in the diagram from this table have corresponding type declarations in the code (lines 2-6).

A type declaration may introduce relations. For example the "Account" type (line 2) introduces four relations (txnList, customer, creditTxns and debitTxns). The keyword *set* indicates multiplicity: it tells that a relation points to a set of elements. For example, for each element of the "Accont" type there is a set of elements of the "Txn" type. To make multiplicity more strict (like "1..*" instead of "*"), multiplicity facts are used (lines 7 - 18).

In this section we gave only the overview of the axiomatic semantics for our framework. You will find more details about it in Chapter 5 where we give an example of an Alloy code for the model of the Simple Music Management System.

4 Example: Bank Account Model

Our method can be better understood through examples. An example that we present in this chapter illustrates how our modeling framework is used to build models. As we have explained in the beginning of this part (Part II), we used our framework to propose an alternative model for the Simple Banking System described in the Ratio Group's white paper [Collins-Cope98] (see the description in the beginning of Part II).

In this example we specify the Simple Banking System in the "Simple Bank Life Cycle" situation that is the broadest situation in our model (see the upper part of Figure 17).

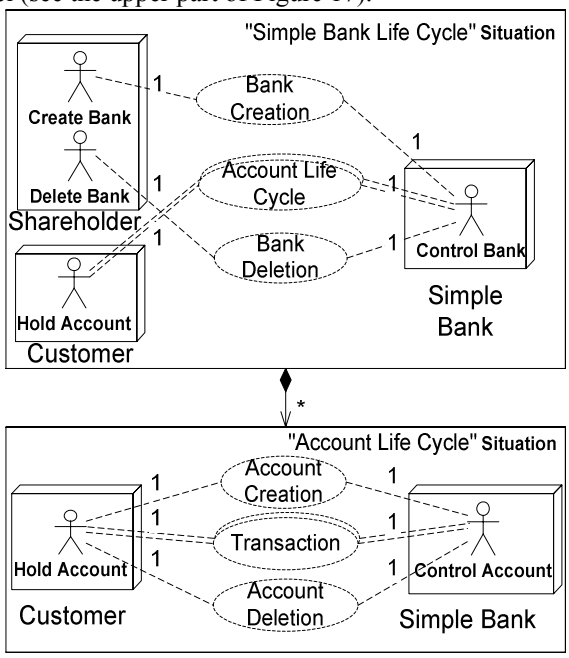
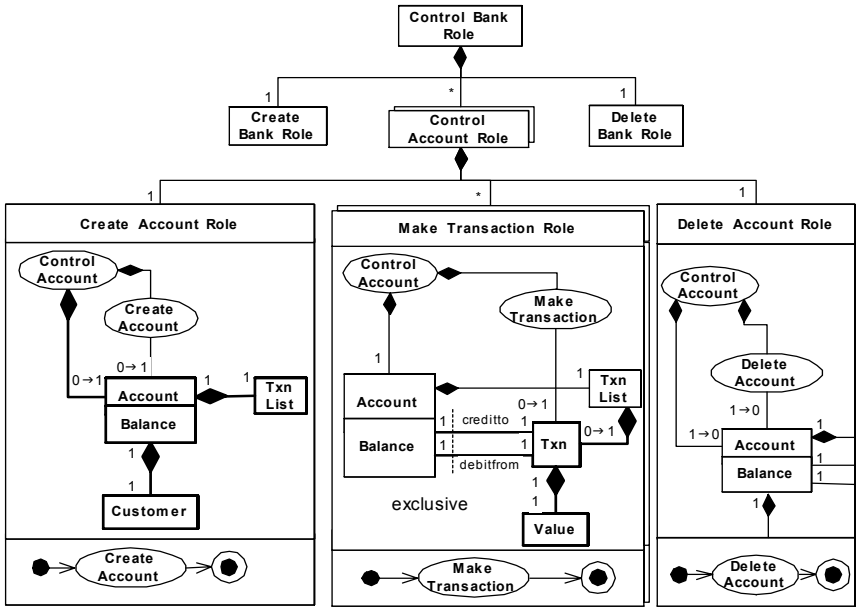


Figure 17. Hierarchy of contexts for a model of the Simple Bank object

4.1 Specification of Base Roles

The "Simple Bank Life Cycle" situation (see Figure 17) includes multiple "Account Life Cycle" situations, one "Bank Creation" situation and one "Bank Deletion" situation. The "Account Life Cycle" situation (lower part of Figure 17) in its turn includes one "Account Creation" situation (see Figure 14 from the previous section), multiple "Transaction" situations and one "Account Deletion" situation. Diagrams from figures 14 and 17 represent the hierarchy of situations for the Simple Banking System. Therefore, up to this point we have satisfied the first two method constraints: we have different situations for the Simple Banking System and we have a hierarchy of these situations.

Now we can build a model of Simple Bank for each of the specified situations (third constraint of our method). We begin with the three lowest-level situations in the hierarchy: "Account Creation", "Transaction" and "Account Deletion". The three corresponding models of a Simple Bank ("Create Account", "Make Transaction" and "Delete Account") in these situations are shown in Figure 18.



$CreateAccount.Account \leftrightarrow MakeTransaction.Account$

$(for \forall MakeTrasaction\ in\ ControlAccount) \leftrightarrow DeleteAccount.Account$

$CreateAccount \rightarrow MakeTransaction$

$MakeTransaction \rightarrow MakeTransaction$

$MakeTransaction \rightarrow DeleteAccount$

Figure 18. "Control Account" role as a set of three lowest-level roles and composition constraints

The situation one level higher in the hierarchy is the "Account Life Cycle" situation. By means of composing the three roles ("Create Account", "Make Transaction" and "Delete Account"), we can obtain the "Control Account" role for this situation. The "Control Account" role is a model of a Simple Bank from the point of view of customer's account. It specifies how a customer works with a single account that he creates in a Simple Bank.

The fourth constraint of our method states that in order to compose roles we have to specify identical attributes and put compositional constraints. We give them in the lower part of Figure 18.

4.2 Composition of Roles

The three models in Figure 18 help a modeler to reason about attributes and actions from roles that are related to the three base situations. However, this diagram does not help a modeler to reason about the "Control Account" role as a whole, without looking into details of the base roles. In order to do this, we use a multi-situational view that we show in Figure 19.

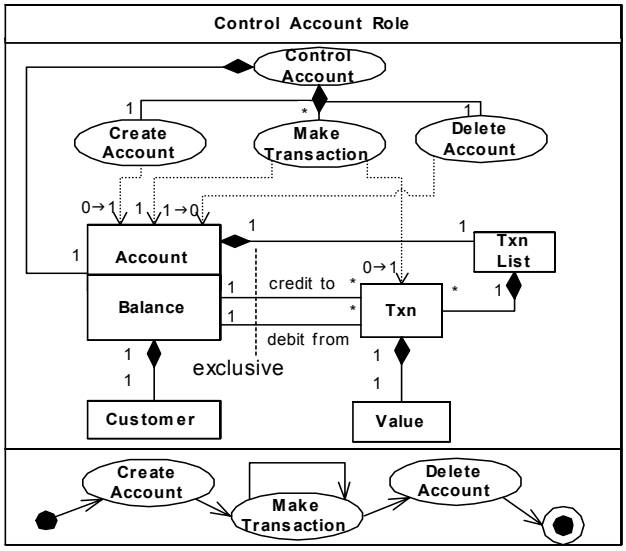


Figure 19. "Control Account" role

A multi-situational view (Figure 19) shows the result of the composition of base roles. An important property of a multi-situational view is that it preserves multiplicity constraints. Multiplicities in Figure 18 are consistent with multiplicities in Figure 19. For example, TxnList in the "Make Transaction" role (Figure 18) includes zero or one Txn attribute because in the context of making a transaction we are interested in the modeling of only one debit/credit transaction. The "Control Account" role includes multiple "Make Transaction" roles (Figure 18). Therefore, TxnList in Figure 19 includes multiple Txn attributes. In another example, the "Make Transaction" role (Figure 18) includes one Account. There are multiple "Make Transaction*s*" roles in the "Control Account" role (Figure 18). However, Account in any "Make Transaction" role is identical with Account in the only one "Create Account" role. Therefore, Accounts in all "Make Transaction" roles are identical and there is only one Account in the "Control Account" role. This kind of reasoning allows for the automatic generation of Figure 19 based on the three based roles and composition constraints presented in Figure 18. The composition of roles as it is shown in this example addresses the scalability problem of visual languages. A visual model can be specified as a composition of smaller visual models (roles) and a composed model (multi-situational view) can be generated automatically.

Let's resume with the specification of the Simple Bank System. Once again we can go to the higher level situation: "Simple Bank Life Cycle". By means of composing the three roles ("Create Bank", "Control Account" and "Delete Bank"), we can obtain the "Control Bank" role for this situation. The composition of the three roles is done in a similar way as at the previous level of the situation hierarchy (therefore, we do not show details of the composition). However, this composition is different in one important aspect: there is a new "Bank Balance" attribute in a model (see Figure 20). This attribute illustrates a property that emerged as the result of the composition (see Section 1.2.2 in Part I where we talked about emergent properties). This property has an important business value. The idea of this attribute is to relate all accounts of a Simple Bank. This can be expressed with the following composition constraints:

ControlBank.BankBalance =

$$\sum_{\text{Control Account}} \text{ControlAccount.Account.Balance} \qquad (1)$$

$$\text{ControlBank.BankBalance} > 0 \qquad (2)$$

These constraints show that the balance of a Simple Bank is the sum of all account balances in this bank. Thus we can see the idea of a Simple Bank: it accumulates money from depositors (customer's credit transaction). A Simple Bank can also invest money (customer's debit transactions) if the overall balance of a Simple Bank is positive (see the second constraint above).

Based on the composition constraints given above, the three roles ("Create Bank", "Control Account" and "Delete Bank") can be composed. The result of composition is given in Figure 20.

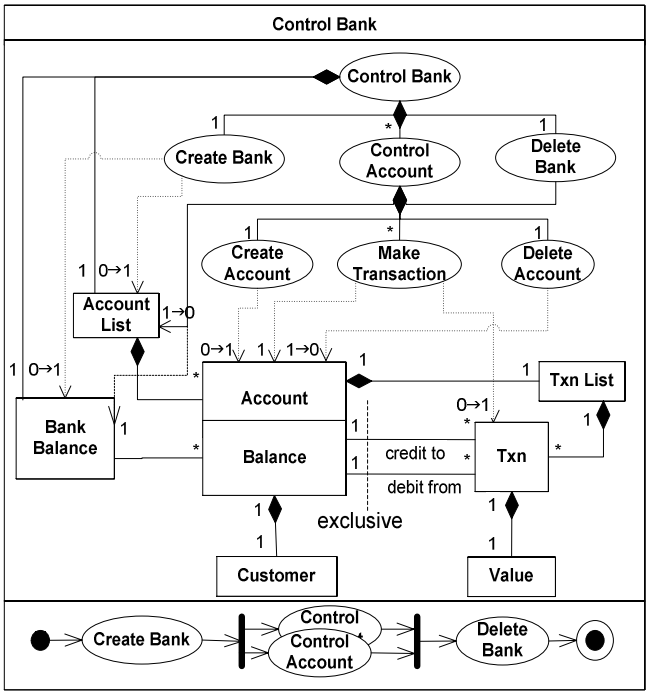


Figure 20. "Control Bank" role

The "Control Bank" role is a model of a Simple Bank that includes several customers' accounts. The goal of this model is to show how a Simple Bank accumulates money from depositors and invest them. This model corresponds to the Simple Bank Life Cycle Situations (see Figure 17).

Putting actions on the same diagram with attributes (or concepts), such as in Figures 19 and 20, provides several advantages for system modeling. It allows for:

- Linking a model (i.e. a role) with a concrete situation for which this model is defined. This helps to avoid diagram misunderstanding such as in the case with the diagram in Figure 12 (see the beginning of Part II). In our approach we explicitly make a choice of a situation that we model. For example, for a Simple Bank, depending on a situation, we can get diagrams in Figure 19 or 20.
- Relating system attributes and behavior. This helps in the analysis of models. Analysis becomes easer because a business modeler can immediately see how actions can influence the state of a system. Therefore a modeler can "play" with a model by mentally "executing" actions. This can be important in discussions with non-professionals (customers, business people).
- Making explicit the life cycle of attributes. Our framework forces a modeler to think about attribute creation, persistence and deletion, which helps a modeler to avoid mistakes in later phases. In case of IT systems, modeling of attribute persistence is useful for its later implementations with data bases.

Note that if we remove all actions from the middle pane of the "Control Account" and "Control Bank" roles, we can obtain the usual UML class diagram for the Control Bank role (Figure 21).

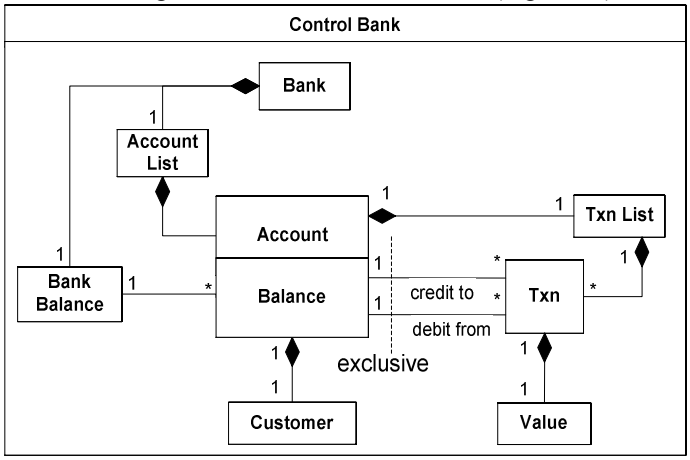


Figure 21. The UML class diagram for the Control Bank role (the Simple Banking System)

The diagram in Figure 21 originates from Figure 12. The problem with the diagram from Figure 12 was that the purpose of this diagram was not clear. It represented some (but not all) concepts that were used to explain two roles of a Simple Bank object ("Control Account" and "Control Bank"). As a result neither of two roles was modeled accurately. In the diagram from Figure 21 we do not have such a problem. It specifies only concepts necessary for reasoning about both roles of a Simple Bank System in the corresponding situations.

5 Example: Simple Music Management System

In this chapter we give an example of a model of Simple Music Management System (a simplified version of www.mp3.com) made with the help of our modeling framework. We took mp3.com as a base of a Simple Music Management System because its model is a complex one and can not be modeled from a single point of view. The goal of this example is to show how an axiomatic semantics can be used to check the correctness of models in our framework.

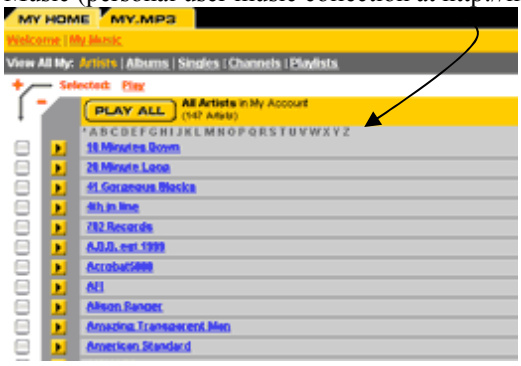
Let's see how axiomatic semantics can be useful in the context of our work. Our framework supports modeling of systems in multiple situations. For each situation we specify a separate role of a system. The whole system is specified as the composition of smaller roles. The reasoning about the whole model as the composition of multiple roles is difficult. It is difficult because the same entity in the Universe of Discourse (UoD or reality) can be modeled as several independent model elements. The role of modeler in this case is to make one model from the set of models corresponding to different situations. A modeler has to find identical model elements (elements that model the same entity in the UoD) in models to be composed and make sure that a resulting model makes sense (it reflects "reality" or the UoD in some meaningful way). In our example we show how to ensure that a composed model gives us some adequate reflection of reality. We evaluate each role as well as the whole model of a system by means of the analysis of the possible instance diagrams (examples). These instance diagrams can be generated automatically based on the Alloy-based axiomatic semantics for our modeling language (see Section 3.3.2 about axiomatic semantics).

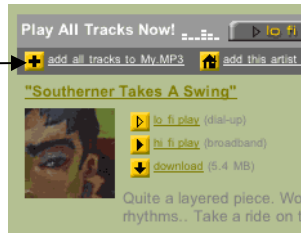
5.1 Simple Music Management System Overview

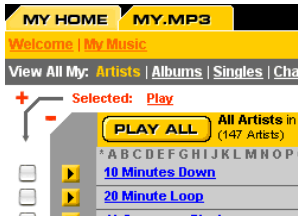
In this example we consider only two major views for this system: User View and Artist View. Here we give a brief description of these two views:

MP3.com users can:

- Find free artist albums and artist singles in places like MP3.com artist pages (pages containing artist albums, singles etc), and add them to My Music (personal user music collection at http://my.mp3.com).







- Play music from My Music.
- Manage personal user music collection (My Music):
  - Manage user play lists. A user play list can include single tracks (a user track that is not part of any other user album) or album tracks from the personal user music collection (My Music).
  - Delete user playlists, singles and albums from My Music.

MP3.com artists can:

- With a Standard Content Agreement:
  - Create one or more Artist Pages where an artist can post his materials
  - Artist materials can include: artist singles and artist albums with album tracks.

In Section 5.2 we begin with the specification of two base roles of a Simple Music Management System: "Manage One User Music" and "Manage One Artist Music" (roles in the lower part of the role hierarchy in Figure 22). The first role is a role of the system from the point of view of one user that manages his music (My Music). The second role is a role of the system from the point of view of one artist that manages his music to be provided for users. These roles are composed into the "Manage Multiple User Music" and "Manage Multiple Artist Music" roles correspondingly. The first one specifies the system in a situation where there are multiple

users managing their music; the second role specifies the system in a situation where there are multiple artists, managing their music.

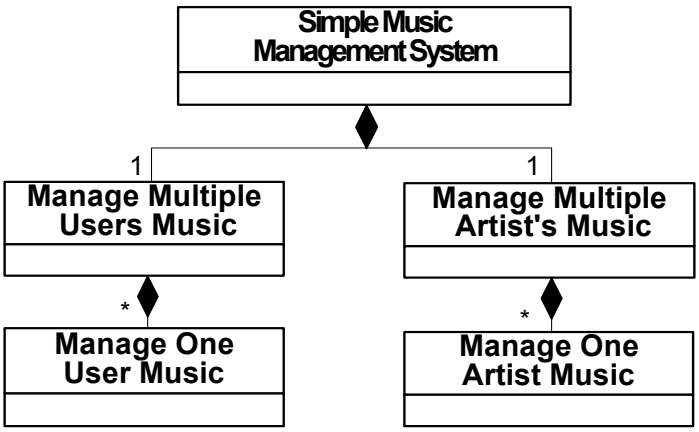


Figure 22 Hierarchy of roles for a model of a Simple Music Management System.

In Section 5.3 we explain how a composition of the "Manage Multiple User Music" and "Manage Multiple Artist Music" roles can be specified. In Section 5.4 we explain how a composed model can be evaluated based on the analysis of automatically generated instance diagrams.

5.2 Specification of Base Roles

In this section we consider modeling and analysis of a role state structure (a collection of attributes, attribute values and relations between attributes). Graphically we represent a role state structure in a similar way to our aforementioned visual language. The main difference with our visual modeling language presented in Section 3 is that we replace ovals that represent actions "Manage X" with the concept "X". For example, Figure 23.a shows an example of the "Manage One User Music" role. In Figure 23.b we replaced the "Manage One User Music" action (form Figure 23.a) with the "User Music" concept. We do such replacements in order to have a uniform mapping of our models to an Alloy-based axiomatic semantics.

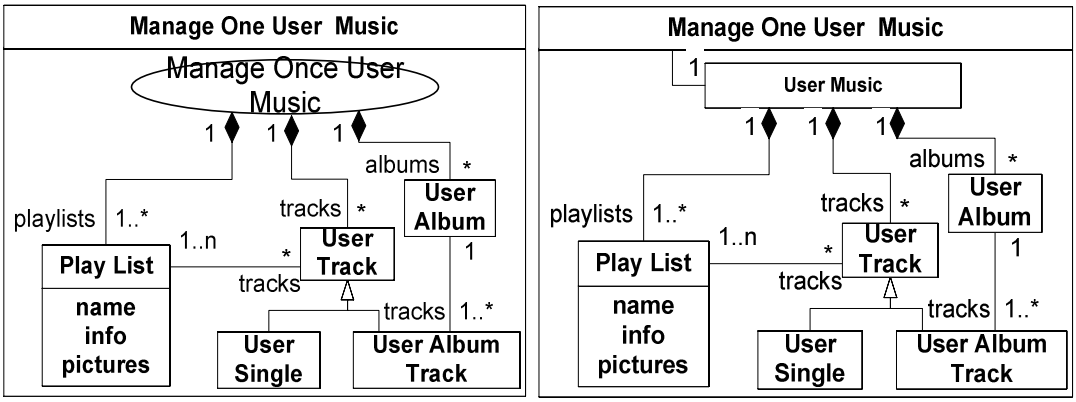
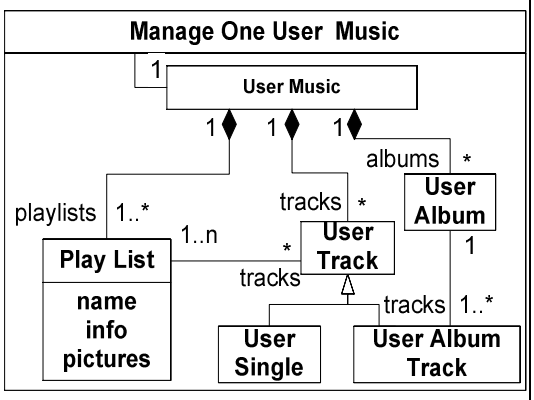


Figure 23. Graphical notation that shows a role state structure for the "Manage One User Music" role.

We also pay your attention that in this section we do not show the third (lower) pane of role models. We remind you that it was used to represent the sequence of actions belonging to a role.

Let's see how the "Manage One User Music" role is mapped to the Alloy code that represents the axiomatic semantics of our visual modeling language (see Table 6).




```
1.  module Main/UserMusic/OneUserMusic   //module declaration
2.  sig UserTrack {}
3.  sig UserMusic { tracks: set UserTrack,
                    playlists: set PlayList,
                    albums: set UserAlbum}
4.  sig UserAlbum {tracks: set UserAlbumTrack}
5.  part sig UserSingle, UserAlbumTrack extends UserTrack {}
6.  sig PlayList {tracks: set UserTrack}

// ------------------ Multiplicity Facts -------------------------
7.  fact {   // PlayList (1..*) -> (*) UserTrack
8.     all ut:UserTrack | some pl:PlayList | ut in pl.tracks }

9.  fact {   // UserMusic (1) -> (1..n) PlayList
10.    (all um:UserMusic | some pl:PlayList | pl in um.playlists) &&
11.    (all pl:PlayList | one um:UserMusic | pl in um.playlists) }

12. fact {  // UserMusic (1) -> (*) UserTrack
13.    all ut:UserTrack | one um:UserMusic | ut in um.tracks }

14. fact {  // UserMusic (1) -> (*) UserAlbum
15.    all ua:UserAlbum | one um:UserMusic | ua in um.albums }

16. fact { // UserAlbum (1) -> (1..*) UserAlbumTrack
17. (all ua:UserAlbum|some uat:UserAlbumTrack|uat in ua.tracks) &&
18. (all uat:UserAlbumTrack | one ua:UserAlbum | uat in ua.tracks) }
// ------------------ End of Multiplicity Facts -------------------------
```

Table 6. State structure model for the "Manage One User Music" role and the Alloy code

Let's look at the Alloy code in Table 6. All rectangles (attributes) in the diagram from this table have corresponding type declarations in the code (lines 2-6). A user track can be either a user single (a track that is not a part of any user's album) or a user album track. In code this is expressed as the partitioning of all user tracks into two sets (line 5).

A type declaration may introduce relations. For example the "User Music" type (line 3) introduces three relations (tracks, playlists and albums). The keyword *set* indicates multiplicity: it tells that a relation points to a set of elements. For example, for each element of the "User Music" type there is a set of elements of the "User Track" type. To make multiplicity more strict (like "1..*" instead of "*"), multiplicity facts are used (lines 7 - 18). For example, the fact in line 16 contains two constraints (line 17 and 18). The constraint from line 17 tells that for all elements of the "User Album" type there are some (at least one) elements of the "UserAlbumTrack" type in the "tracks" relation.

As we mentioned in the introduction, the simplest way to make an agreement with no professionals (like customers) that a model adequately represents the UoD is to consider model instances. Model instances can be generated using the Alloy Constraint Analyzer. It checks the consistency of a formal Alloy model; randomly generates a solution (an instance of a model) and visualizes it. In order to generate a solution, a scope of a model should be defined. A scope defines how many instances of each type a solution may include.

Examples from Figure 24 and 25 were built with the Alloy Constraint Analyzer.

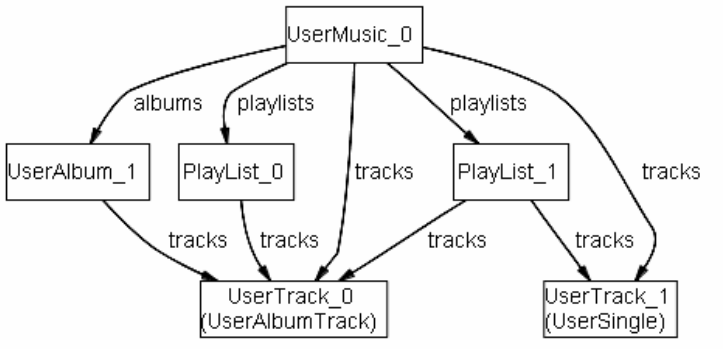


Figure 24 Example with a scope 2 but 1 UserMusic

The scope of the example in Figure 24 was defined in the following way: 2 All, 1 UserMusic. This means that this example, generated by the Alloy Constraint Analyzer, could have maximum two elements of each type (except the UserMusic type) and one element of the UserMusic type. Having only one element of the UserMusic type corresponds to the fact that the "Manage One User Music" role includes one UserMusic attribute (see

diagram in Table 6). The Alloy Constraint Analyzer allows for the generation of several different examples for the same model and the same scope. All these examples can be used to reach an agreement with a customer if the state structure model of the "Manage One User Music" corresponds to the common understanding of the UoD.

The next step will be to make a state structure model for the "Manage Multiple User Music" role (see figure in Table 7). Comparing with the diagram from Table 6, we changed the multiplicity of the association that connects the "Manage Multiple User Music" role with the UserMusic attribute (see figure in Table 7). This allows for specifying that this role has multiple "User Music" attributes. All other elements of the diagrams from Tables 6 and 7 are the same (for the moment we ignore *inv1* and *inv2* in the diagram from Table 7). This allows us to reuse the Alloy code from Table 6 and test it with a new scope. With a new scope we have to require that a solution includes several (at least two) instances of the UserMusic type. Therefore we choose a new scope equal to two. This means a solution may include maximum two instances of each type.

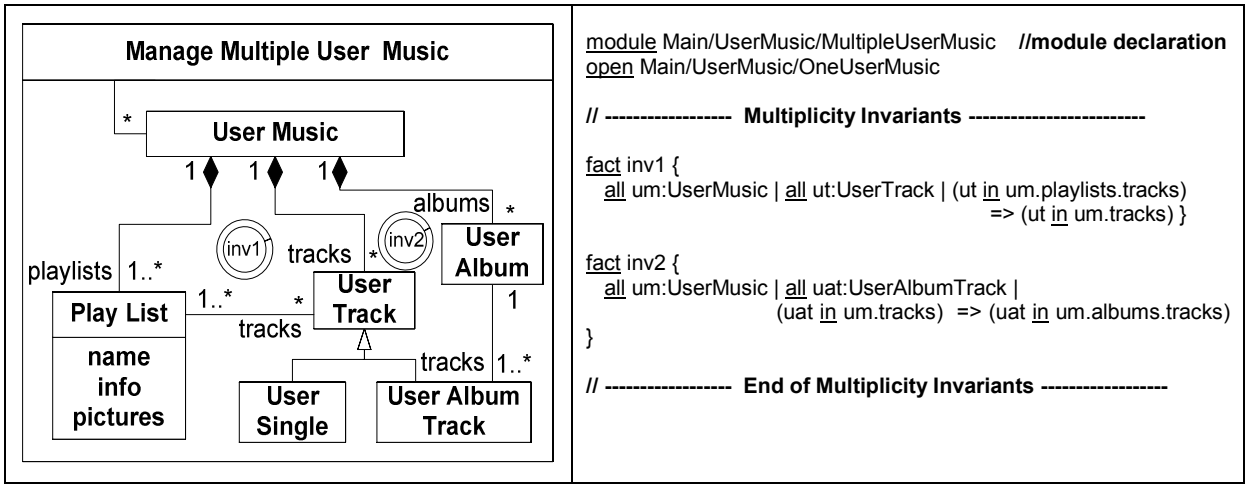


Table 7 Specification of state for the "Manage Multiple User Music" role and the corresponding Alloy code.

A solution, found by the Alloy Constraint Analyzer for the Alloy code from Table 7 with the scope equal to two, is shown in Figure 25.

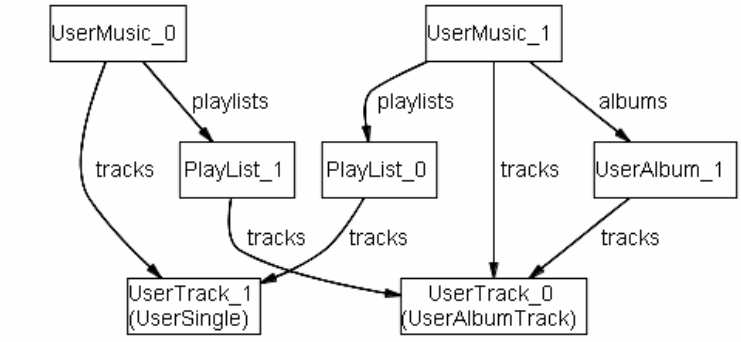


Figure 25 Example with a scope 2

We can see that this solution does not adequately show the reality: "A user play list may include single tracks or album tracks from this user music" (see the description of a Simple Music Management System in introduction). However, the solution, found by the Alloy Constraint Analyzer, shows that a user play list may include tracks of another user. Therefore we have to put additional constraints to make a state structure model for the "Manage Multiple User Music" correct. These additional constraints we put in a separate Alloy module: MultipleUserMusic (see Table 7).

Additional constraints come directly from the diagram in Table 7: for all cycles in this diagram we add invariants in the Alloy code. For example, *inv1* requires for all users, that tracks from all user playlists (for a given user) are tracks of the same user. The MultipleUserMusic Alloy module can be analyzed by means of generating different examples. We do not give them here since they are similar with examples from Figures 24 and 35.

Based on the description of our small case study, we can build a state structure model of the "Manage Multiple Artist Music" role (see Figure 26).

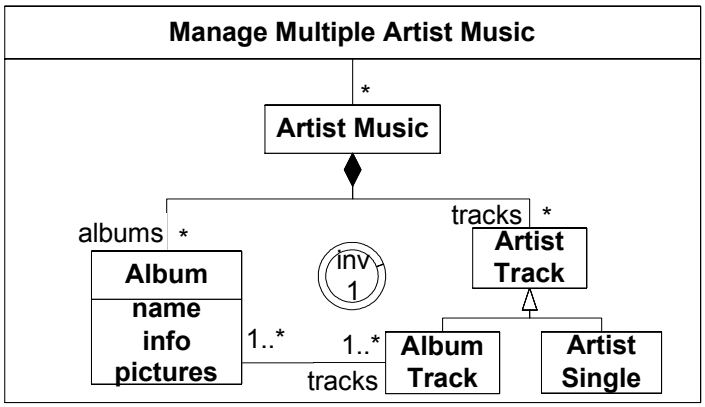


Figure 26 The state structure model for the "Manage Multiple Artist Music" role.

This model and the corresponding Alloy code are built in the same way as for the "Manage Multiple User Music" role. Therefore we do not show the Alloy code here.

Let's conclude: up to this point we have the role state structure models for both "Manage Multiple User Music" and "Manage Multiple Artist Music" roles. In the next section we show how a composition of these two models is done.

5.3 Composition of Roles

As we explained earlier, the modeling process of our modeling framework consists in building models for specific situations and in finding identical elements in these models. In this section we continue the example of a model of a Simple Music Management System where we have identified some identical elements. Let's imagine an external observer who perceives the work of two users (see Figure 27).

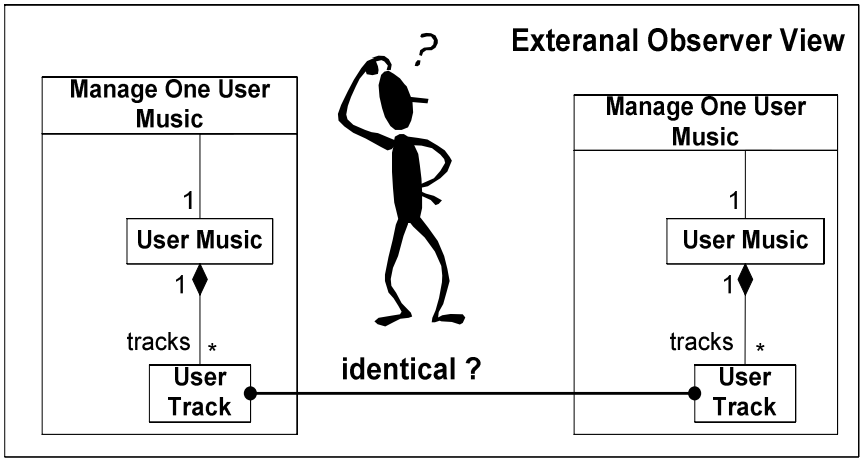


Figure 27 Identical model elements.

We suppose that this external observer can not see all details about user tracks: how they are created, used and deleted. The only one thing that he can observe is the content of tracks. This defines an External Observer view. From this point of view, the external observer may say that some user tracks of some different users are identical, i.e. the music of these tracks is the same. The question is: if we have to model this identity or not. In our case study, this identity is important because it takes into consideration the Artist Track concept from the "Manage Multiple Artist Music" role: two tracks can be considered identical if they correspond to the same artist track.

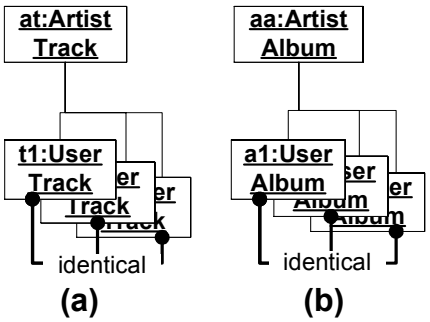
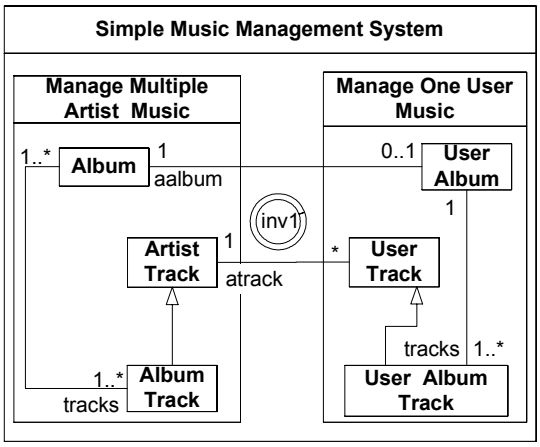


Figure 28. Modeling of identical model elements: (a) Identical User Tracks; (b) Identical User Albums

To model the identity of model elements, we define a new attribute (concept) that is associated with all identical elements. In our case study, this new attribute is the Artist Track attribute from the "Manage Multiple Artist Music" role (see Figure 28.a). Figure 28.a shows that an Artist Track has multiple identical User Tracks. In the same way as User Tracks can be identical because they refer to the same artist track, user albums can also be identical because they refer to the same artist album. Therefore, we model that an Artist Album has multiple identical User Albums (see Figure 28.b).

Let's conclude. We modeled the state structure of two roles ("Manage Multiple User Music" and "Manage Multiple Artist Music"). We also modeled how the state structures of these two roles are composed (using a diagram from Figure 28). All these models give us the state structure model of a Simple Music Management System. At this point we can create an Alloy code (see Table 8) that reflects the composition of the two roles and then we can do the analysis of the complete model.




```
1.  module MusicManagementSystem   //module declaration

2. open MusicManagementSystem/UserMusic/MultipleUserMusic
3.  open MusicManagementSystem/UserMusic/OneUserMusic
4.  open MusicManagementSystem/ArtistMusic/MultipleArtistMusic
5.  open MusicManagementSystem/ArtistMusic/OneArtistMusic

7.  sig UserTrack1 extends UserTrack {atrack: ArtistTrack}
8.  sig UserAlbum1 extends UserAlbum{aalbum: Album}

9.   fact { all ut:UserTrack | ut in UserTrack1 }
10. fact { all ua:UserAlbum | ua in UserAlbum1 }

// ------------------  Multiplicity Invariants -------------------------
11. fact {      // Multipl: Album (1) <- (0..1) UserTrack
  all  um:UserMusic  |  all  a:Album  |  sole  ua:um.albums  |  a  =
ua.aalbum }
// ------------------  End of Multiplicity Invariants ----------------

// ------------------  Multiplicity Invariants -------------------------
12. fact inv1 {  all um:UserMusic | all ua:um.albums |
                  ua.aalbum.tracks = ua.tracks.atrack }
// ------------------  End of Multiplicity Invariants ----------------------

13.  fact {         // Definition of UserSingle: a user track
                    // that is not a part of any user album
   all um:UserMusic | all us:UserSingle |
   (us in um.tracks) => (us.atrack not in um.albums.tracks.atrack) }
```

Table 8 State Structure Composition for the "Manage Multiple User Music" and "Manage Multiple Artist" roles.

We have to reflect in the Alloy code new relations between the AtristTracks and UserTrack types and between the Album and UserAlbumTrack types. To make a new relation between two already specified attributes in the current version of Alloy, we have to extend existing types: UserTrack and UserAlbum (lines 7-8 in Table 8). To ensure that new attributes (UserTrack1 and UserAlbum1) and their predecessors participate in the same relations we created two facts (line 9-10 in Table 8). Line 11 in Table 8 specifies the multiplicity invariant. Note that this multiplicity is specified in the context of one user (as it is shown in a diagram from Table 8): we require that for any artist album there is only one user album in the context of a given user music. When we create an Alloy code, we have to take into account possible "conceptual cycles". In the same way as in the previous examples we have to create invariants for these cycles.

The last constraint[22] that we add to the Alloy code is related with the definition of a user single track: a user track that is not part of any other user album (see introduction). The fact from line 13 specifies this constraint. It tells that for any user single of some given user, the artist track for this user single is not an artist track for any album track for the same user.

5.4 Analysis of the Composed Model

Based on the Alloy code from Table 8, several instance diagrams can be generated. One of these diagrams is shown in Figure 29.

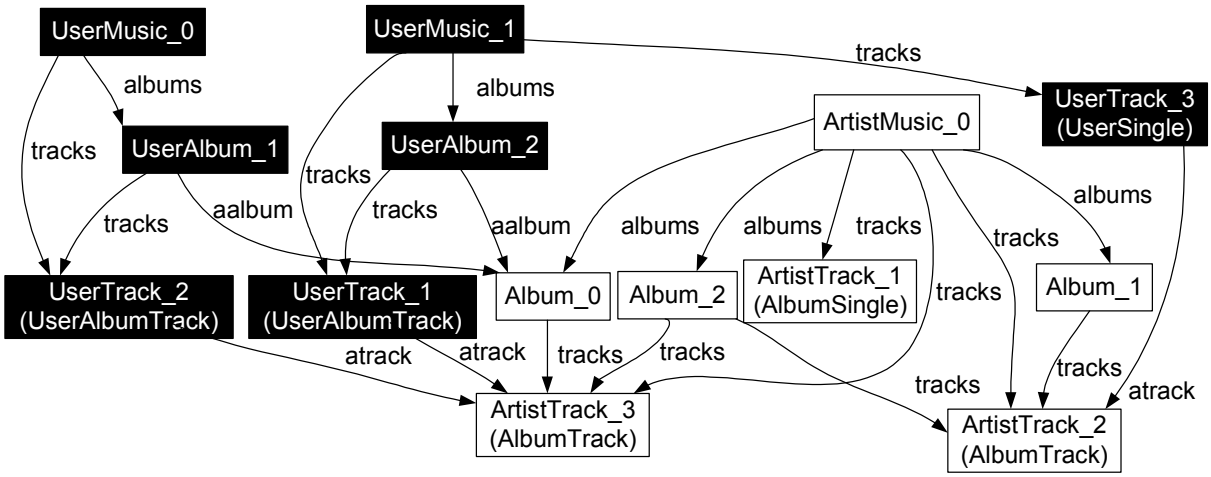


Figure 29. Instance diagram for the Simple Music Management System.

We used filters provided by the Alloy Constraint Analyzer tool to hide instances of the User Play List attribute. This attribute is not involved in the composition of base roles and therefore is not necessary for the analysis of the composition.

To make the analysis easier we colored the instances of attributes from the "Manage Multiple User Music" role in black. All other instances of attributes are from the "Manage Multiple Artist Music" role. The diagram from Figure 29 can be used for the analysis of the model of a Simple Music Management System. This analysis can be done in collaboration with a customer who asked for the development of this system. The goal of the analysis is to show possible states of the system (instance diagrams) and ask if these states can adequately represent the reality. For example, the following question can be asked: If a user single track (see UserTrack_3 in Figure 29) can be related with an artist album track (see ArtistTrack_2 in Figure 29), i.e. if a user can add to its music collection only one track from an artist album versus adding the whole album? Similar questions can be asked based on several instance diagrams such as one in Figure 29. This kind of model analysis helps for avoiding mistakes that can be discovered later in the implementation phase.

Part II Summary

In this part of our thesis we have defined our Situation-Based Modeling Framework. This framework is based on a set of epistemological principles that we have identified in the state of the art analysis part (Part I). The following three epistemological principles we consider as the most important in the context of our work: situation relativity, state and behavior holism and the principle of diagrammatic representation. The Situation Relativity is the central principle of our modeling framework. In order to make situations the first-class citizens in our modeling framework, we used the Situation Theory as the foundation of our framework. The explicit modeling of situations makes models more systemic and, therefore, makes system stakeholders' (users, developers, etc) reasoning about models easier.

We have defined our framework based on the three directions of Systems Inquiry:

- Framework philosophy: we have defined the main concepts of our modeling framework and the modeling language that is based on these concepts;

[22] This constraint was discovered in the result of the analysis of instance diagrams generated by the Alloy Constraint Analyzer.

- Framework theory that explains how a system is specified as the composition of the system roles;
- Framework methodology that explains how models are built using our modeling language; the method gives a set of constraints that a modeler should satisfy in the design process. As a part of framework methodology, we have also defined the axiomatic semantics for our modeling language. Axiomatic semantics allows a modeler to check the correctness of models by means of defining the set of invariants on the state of a system.

To illustrate the modeling language and the method of our framework we have used an example of the Simple Banking System. This example shows how a model of the Simple Banking System is specified as the hierarchy of situations and the corresponding roles. To illustrate the axiomatic semantics, we have given the small case study of a Simple Music Management System where we have shown how the axiomatic semantics is used to check he correctness of models built with our framework.

Our framework can be applied at any given level of an organization hierarchy. It allows systematically define the functionality of computational objects at each level. The relation between deferent organization levels, i.e. how computational objects are refined between organization levels, is left out of the scope of our framework. However, in order to improve the overall understanding of our framework, in Appendix 1 we gave some basic ideas about the refinement of computational objects. These ideas allow for making a bridge to the technologies that can be used for the implementation of models built with out framework. In Appendix 1 we have given general suggestions that can help designers to choose the appropriate methods for the implementation of our role models. In these suggestions we propose two implementation patterns for the composition of roles and the set of corresponding methods for each pattern.

PART III

APPLICATION OF THE SITUATION-BASED MODELING FRAMEWORK TO SOFTWARE ENGINEERING AND BUSINESS PROCESS MODELING

Overview: *In this part we describe the application of our framework in two domains: software engineering and business process modeling. In the domain of software engineering we used our framework as the basis for the Rapid Prototyping Tool based on Aspect-Oriented and Subject-Oriented programming (Chapter 6). In the domain of business process modeling we applied our framework as the theoretical background for the conceptual tool for business process innovations (Chapter 7) and for the specification of a manufacturing process (Chapter 8). The last application was done in collaboration with an industrial partner and serve as a practical validation of our framework.*

6 Application to Software Engineering

In this chapter we give an example of the application of our framework in the field of software engineering.

In our framework we specify systems by means of considering them in different situations. For each situation a modeler specifies base roles and then composes them into the whole model of the system. The analysis of models that integrate multiple roles, however, is difficult. In many cases models can be evaluated only after their implementation. Building a rapid prototype for the system, specified as the composition of multiple roles, can be helpful for evaluating system's key features before the implementation. We understand rapid prototyping to be an early development phase for building small-scale implementations (prototypes). In this section, we present a tool that supports the role composition. It is based on two programming techniques: Subject Oriented Programming (SOP) and Aspect Oriented Programming (AOP).

6.1 Introduction

In the early stages of software development, it is important to reach an agreement between system developers and customers (or other system stakeholders) about system requirements. This agreement can be reached by means of specifying system requirements in the form of a model that can be discussed with all stakeholders.

Modeling complex systems poses a problem: models are very large and hence difficult to understand. The solution to this problem is to make models by composing small roles, where each role is small enough to reason about: each role can be discussed between stakeholders and then the composition of roles needs to be performed. The composition, however, raises a new problem: the analysis of models that integrate multiple roles is difficult. In many cases such models can be evaluated only after their implementation. Building a rapid prototype for a system defined in multiple situations can be helpful in evaluating key features of a system before its final implementation. "At very early stages of planning, a small-scale prototype is built that exhibits key features of the intended system. This prototype is explored and tested in an effort to get a better handle on the requirements of the larger system. This process is called Rapid Prototyping" [Wilson93]. Rapid prototyping allows system stakeholders to reason about and test the functionality of the future system. A prototype guarantees that system's requirements correspond to the common understanding of all stakeholders.

In this section we use our situation-based modeling framework to improve the early requirements specification process by means of a situation-centric design method and corresponding rapid prototyping tool. Our framework is based on the visual language that gives us an opportunity to overview the structure of a system and allows us "…to discuss and validate process models with both users and owners of the process, many of whom are not prepared to invest their time in understanding more complex representations" [Phalp97]. The main advantage of our visual language is human orientation: it improves the process of understanding models by humans. It is based on the philosophical and psychological principles that support human reasoning.

One of these principles is that every role of a system is specified in its own *situation*. The situation is modeled as a set of collaborating roles. To reason about a system as a whole, one should analyze relationships between different roles of the system. The notion of *compositional constraints* is used to show how different roles of a system are composed into bigger roles. In the composition process we understand *role composition* as (1) linking the *identical* model elements, representing the same entities in the Universe of Discourse, and (2) putting *constraints* between model elements found in these roles. Besides this, we also consider placing the *multiplicity* on roles as an important operation when composing a larger role out of several small ones.

We developed an archetype of the Rapid Prototyping (RaP) tool that supports our method. The RaP tool is based on the same idea: prototyping small (base) roles and composing them into a prototype of a system. We used Java to make prototypes of base roles and the subject oriented programming (SOP) together with aspect-oriented programming (AOP) to implement the composition of base roles.

We continue do describe the application of our framework in the field of software engineering in the following way: in Section 6.2 we describe how our Situation Based Modeling Framework can be supported by RaP in the context of software engineering. In Section 6.3 we illustrate RaP with an example. In Section 6.4 we make a summary of main results.

6.2 Technology Leveraged by the Rapid Prototyping (RaP) Tool

As we discussed in above, the use of prototypes can improve the quality of requirements and makes the agreement between system's stakeholders easier. Furthermore, a prototype can be used as a basis for a final implementation. In this section we explain how a Rapid Prototyping (RaP) tool can be built to support the generation of prototypes for role-based models.

Prototypes can be built similarly to the way models are built in our modeling framework: system designers have to begin with the specification of roles (mapped to base situations) and by writing a corresponding code; then these roles are composed into larger specifications and a corresponding composed code.

There are several case tools that support the generation of code from models (like Rational XDE, Telelogic TAU, Borland Together etc.). Therefore, in our work we do not explain a code generation process for base roles. We suppose that each role at the lowest level of a role hierarchy is simple enough. Hence it is simple to generate a code from it. In our work we explain how the composition of code for base roles can be performed based on the identity and compositional constraints that we have defined in our framework.

There are several technologies that allow for the composition of code fragments (such as AOP [Kiczales97], SOP [Harrison93], role components [VanHilst96] or coordination contracts [Andrade99], etc). In our work we decided to use only two of them: SOP and AOP. We found the SOP idea to be very similar to the role composition in our framework: they both specify the identity of model elements. The AOP aspects seem suitable for implementing composition constraints. Furthermore, there are software tools for both SOP and AOP approaches based on Java. Therefore, we use Java as a basis for prototype implementations.

The overall procedure of building system prototypes based on the design method of our framework is the following:

1. Specify base roles at the lowest level of a role hierarchy using our visual modeling language;
2. Generate Java code for each base role;
3. Compose base roles by putting composition constraints and adding multiplicities,
4. *Automatically* produce a corresponding composition of the Java classes at the implementation level. This means: generate Java code that reflects the role composition to provide a prototype implementation;
5. Use the generated prototype implementation to test a model;
6. Improve the model by changing the rules of a composition, adding constraints or modifying base roles. Repeat steps 3-6.

Based on this process, we describe a prototype of the RaP tool and explain how SOP and AOP are used to support our approach.

6.2.1 Identity implementation with SOP

Subject Oriented Programming was proposed by Harrison and Ossher (from IBM) as an extension of the object-oriented paradigm to address a problem of handling different subjective perspectives on objects to be modeled. According to [Harrison03] the term *subject* means a collection of state and behavior specifications reflecting the perception of the world. SOP uses them to represent a subjective view on objects. Any object can be seen as a composition of several subjects, where each subject can be managed separately.

The description of roles in our modeling method is closely related to SOP subjects. In turn, the SOP composition procedure is similar to the composition of roles in our framework, i.e. it allows for implementing model elements identity. Thus we have chosen the SOP approach for a role composition. For testing purposes we use Hyper/J tool [IBM03]. Hyper/J [Tarr00] supports a *multidimensional separation of concerns*[23], an extension of SOP. Hyper/J tool is a standard Java application that allows for the composition of conventional Java classes according to composition rules. The input of Hyper/J is compiled Java class files with a special options file and produces Java class files. The options file indicates which files participate in a composition, how equal named parameters or actions should be treated, and other complimentary information [Tarr00].

Figure 30 shows a composition of roles with a possible Hyper/J implementation. We assume that *CRole1* and *CRole2* classes are implemented in Java and correspond to their visual form (Figure 30.a). The *CRole12* class is a result of composition of CRole1 and CRole2 classes based on a Hyper/J options file. If we compare Figures 30.a and 30.b, we find out that composition constraints (links between identical model elements) correspond to the Hyper/J options file.

[23] The multidimensional separation of concerns introduces a new unit of modularization, other than a class. Each concern encapsulates a particular area of interest [Tarr00].

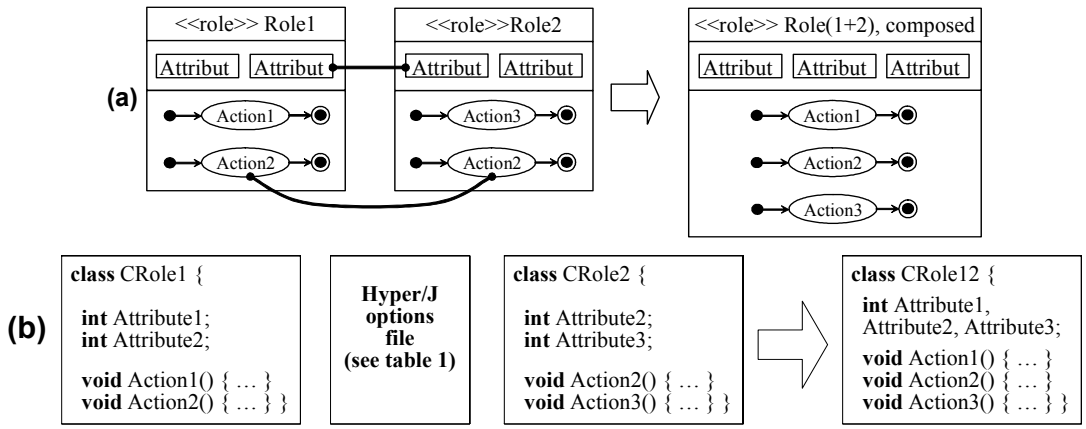


Figure 30. Composition of roles: (a) identity modeling; (b) Hyper/J implementation

In this thesis we give you an overview of the options file. In Table 9 we show the part that is responsible for the compositional constrains.

Table 9. Hypermodule[24] part of the Hyper/J options file

```
Hyperslices: Role.Role1, Role.Role2
 ; define two roles as hyperslices
Relationships OverrideByName;
 ; It specifies what to do if two ore more actions with equal names
 ; are encountered: if OverrideByName – first action substitutes
 ; others; MergeByName ensure consequent invocation of all actions.
Equate class Role.Role1.CRole1, Role.Role2.CRole2
 ; Class CRole1 from Role1 role is equal to class CRole2
Rename class Composed.CRole1 to CRole12
 ; Rename composed class to CRole12
End hypermodule;
```

In order to support our approach in the RaP tool we need to generate a Hyper/J options file automatically from the visual composition links.

6.2.2 Constraint Implementation with AOP

The Aspect Oriented Programming (AOP) paradigm introduces a new concept called *Aspect* for encapsulating a crosscutting code. According to [Czarnecki00] a model is an aspect of another model if it crosscuts its structure. Compared to OOP, an aspect in AOP can be considered as a modular unit like a class. It wraps a supplementary code and also stores information about *join points* and *pointcuts*[25]. Such join points indicate when the supplementary code should be executed. In order to compose original classes with aspects, a waving procedure has to be made.

We use AspectJ 1.1 tool in our experiments. In contrast to Hyper/J, which proposes a composition of pure Java classes according to the options file, AspectJ supports only an augmentation of Java classes with aspects. We found this property convenient for *constraints implementation*. Since AspectJ permits the substitution of a certain method, in the simplest case it can be used to block execution of a certain operation accordingly to a given conditions. Therefore we define constraints as "*execute this operation only if <Boolean expression>*". We believe that such simplified constraint definition covers the majority of the practical cases.

To support role composition, we provide a library of different constraints. Some of them (such as the sequential constraints) can be defined using templates (see Table 10).

Table 10. Constraint implementation pattern for sequential constraints

```
Aspect SomeConstraint {
 Public int flag=0;
```

[24] *Hyperslice* is a set of concerns that is declaratively complete. In turn *hypermodule* comprises a set of hyperslices being integrated [Tarr00].

[25] *Join points* are well known points in the dynamic execution of a program. And *pointcuts* are sets of join points.

```
        pointcut P1():call(* CRole12.Action1(*))
        after():P1() { flag=1; }
        //This code is executed after each call of the Action1 action from CRole1 class

        void around() execution(* CRole12.Action2(*)) {
        // Check flag value
        proceed(); //execute an original action if a condition is fulfilled
        // …
        } //This code is executed instead of each execution of the CRole12.Action2
       }

       . . .
 }//This code is executed instead of the execution of CRole12.Action2
}
```

In this aspect we declare the *flag* variable and control an execution of the *Action2* according to the flag value, changed after *Action1* call. AspectJ compiler allows for compiling (waving) such aspect(s) along with basic classes in order to expand a functionality, i.e. add some constraint.

6.2.3 Multiplicity Implementation

In the composition process, a modeler often needs to put a certain base role into a collection, i.e. the multiplicity for this role should be specified (Figure 31). How can we implement the role multiplicities? We do not go into details about multiplicity implementations - this requires considering many technical aspects. We simply give some suggestions and in Section 3 we explain how we implemented the multiplicity in our example.

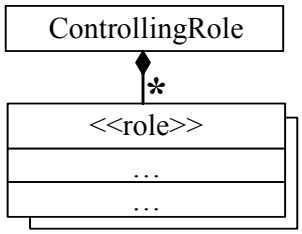


Figure 31. The multiplicity example

We suggest using an additional Java class (ControllingRole) that implements a *list* of base role objects (Table 11).

Table 11. Implementation of the list of base roles

```
public class ControllingRole {

public List RoleInstancesList=new ArrayList(); //List of objects

 //…
 //Implementation of methods.

 //Method names are taken from an original role class.
 //Code for methods is written manually or taken from templates.
}
```

This class will typically have the same methods as a base role. The purpose of this class is to redirect a method invocation to a certain member (role) in the role list. The following (non-exhaustive) choices are possible:

- Create a new member and invoke a corresponding method from this member;
- Invoke a corresponding method from every member of the list;
- Find a certain member by some condition and invoke a corresponding method from that member. It is also possible to modify a signature of a certain method to pass a needed information as a parameter(s);
- Delete the member from the list.

We emphasize one more important property of role multiplicities: it is possible to put a constraint on the number of roles in the role list. This task is very practical and allows for reasoning about the capacity of systems.

6.3 Presentation of the RaP tool

In this section we give an example that shows how the method of our modeling framework (based heavily on role composition) can be supported with the RaP tool. For this example we take a very simple case that shows how a prototype of a Reseller Shop can be built and tested based on our modeling method. The idea is to specify a set of very simple roles with the corresponding Java code (that can be generated using Together or Rational case tools). Based on these roles (and the corresponding code), a prototype of a model can be built by means of composing these roles. The method will navigate a system's developer in the development process by advising roles that should be composed and the way the composition has to be done. In order to see how our modeling method can be supported with the RaP tool, we briefly discuss a model of the Reseller Shop and explain how a prototype of this model is built.

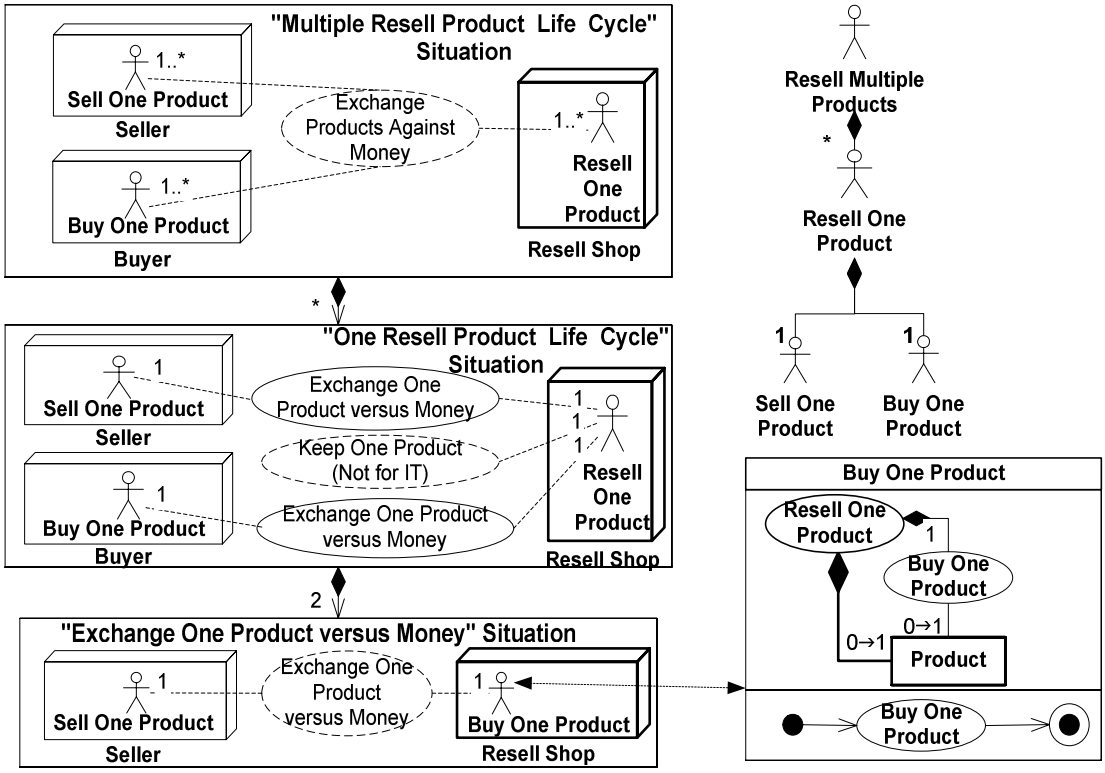


Figure 32. The Resell Shop model

Figure 32 illustrates the main idea of the Reseller Shop model. At the lowest level we have two base roles: "Sell One Product" and "Buy One Product". Both these roles are defined in the "Exchange One Product versus Money" situation. One level higher, these roles are composed into the "Resell One Product" role from the "One Resell Product Life Cycle" situation. On the next level the "Resell Multiple Products" role is a composition of multiple "Resell One Product" roles. This role is defined in the "Multiple Resell Products Life Cycle" situation.

Now we describe a scenario that shows how the RaP tool can be used to generate a prototype for the *Resell* Shop example (more detailed demonstration see in [Skobeltsyn03]). The scenario starts with the input of the base roles *Buy One Product* and *Sell One Product* into the tool. We do not consider how this can be done. For the simplicity, in the RaP tool we call these two roles *Buy* and *Sell* (Figure 33).

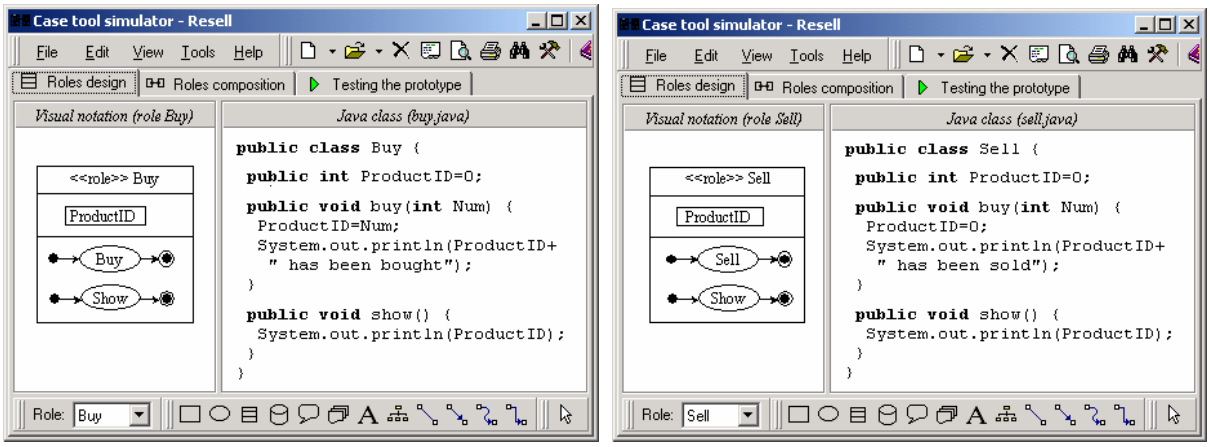


Figure 33. RaP tool: Buy and Sell roles

Let's now consider the following steps of our scenario:

1. Roles composition. The first step in our scenario is to perform the composition of the *Buy* and *Sell* roles (Figure 34). It is done by selecting identical attributes (actions) and linking them. Both, the visual representation of the composed *Resell One Product (*or simply *Resell)* role (Figure 35) and Hyper/J options file are generated automatically. A Java class corresponding to the *Resell* role is produced by means of Hyper/J tool (*buy* and *sell* classes along with the automatically generated *options file* are compiled by Hyper/J compiler). The result of this composition can be consulted by selecting the "Composition result" tab. It is similar with the Resell role in Figure 35.b.

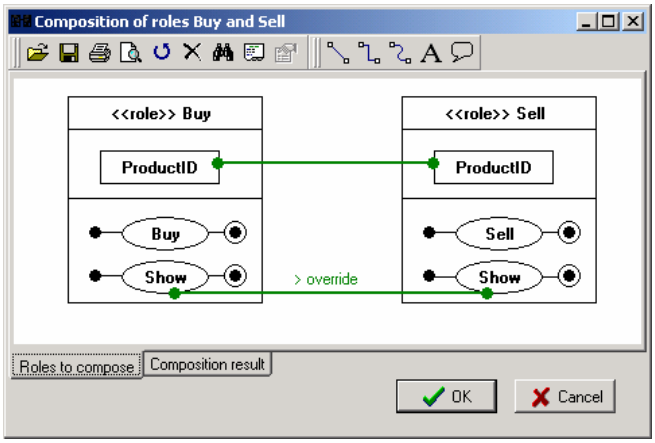


Figure 34. RaP tool: roles composition

2. Constraint placing. The RaP tool offers the following functionality: a user selects an action that should be affected by a constraint. Then the list of available constraints with their properties will drop out (Figure 35.a). In our example we require the *sell* action (from the *Resell* class) to be executed after the *buy* action. We use a *SequentialConstraint* to specify this. The aspect code, stored in the library of constraints, is added to the project by means of AspectJ compiler. A user only selects the names of actions that should be sequentially ordered (Figure 35.a). Figure 35.b shows the result of the constraint placing and Table 12 shows the corresponding code.

Table 12. SequentialConstraint aspect code automatically added to the project

```
aspect SequentialConstraint {
 public int FirstWasExecuted=0;
 pointcut First():call(* Resell.buy(*));
 after():First() { FirstWasExecuted=1; }
 void around():execution(* Resell.sell(*)) {
  if(FirstWasExecuted==0){System.out.println("CONSTRAINT OCCURED"); return; }
  proceed(); //Execute sell() action
  FirstWasExecuted=0; } }
```

a)

b)

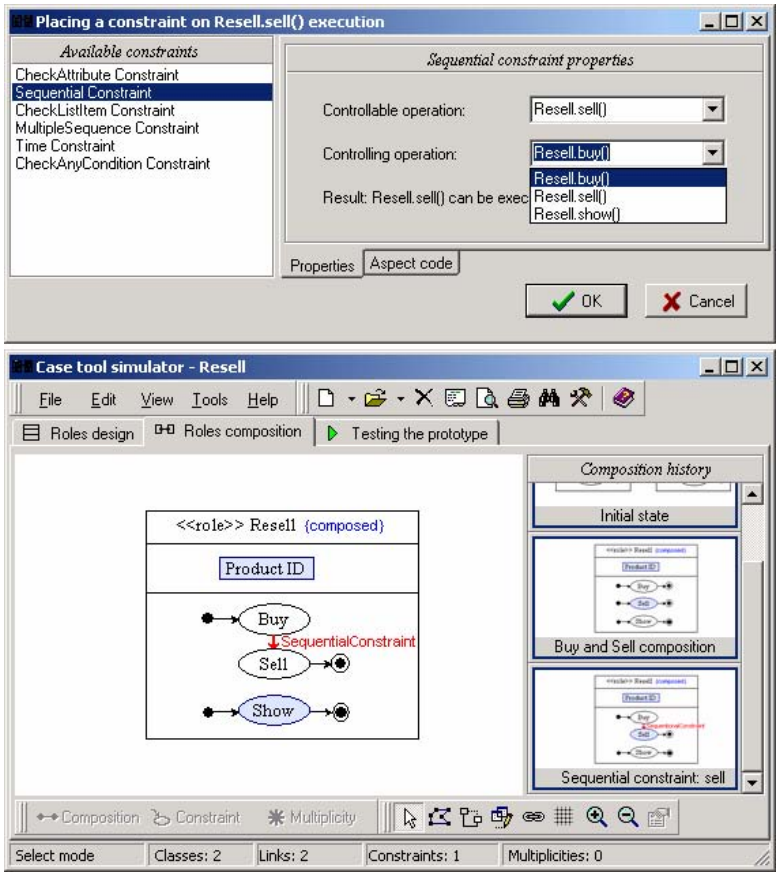


Figure 35. Constraint applying: (a) selecting a constraint; (b) SequentialConstraint applying result

3. Multiplicity. The next step is to implement the *multiplicity* of roles. The multiplicity is based on the role composition that includes roles of the same type. There are different ways of composing the behavior of these roles. For example, we can compose the behavior of multiple Resell roles sequentially. This means that a reseller of multiple products will be able to resell products sequentially. Another way to compose multiple roles of the same type is to compose them in "parallel". In our example, this means that a reseller of multiple products will be able to resell multiple products simultaneously. The "parallel" composition can be implemented also in different ways (see [Skobeltsyn03] for parameters that allows for different "parallel" compositions). In our work we do not discuss different implementations of *multiplicities.* We only explain how the "Resell Multiple Products" role was implemented in our particular case.

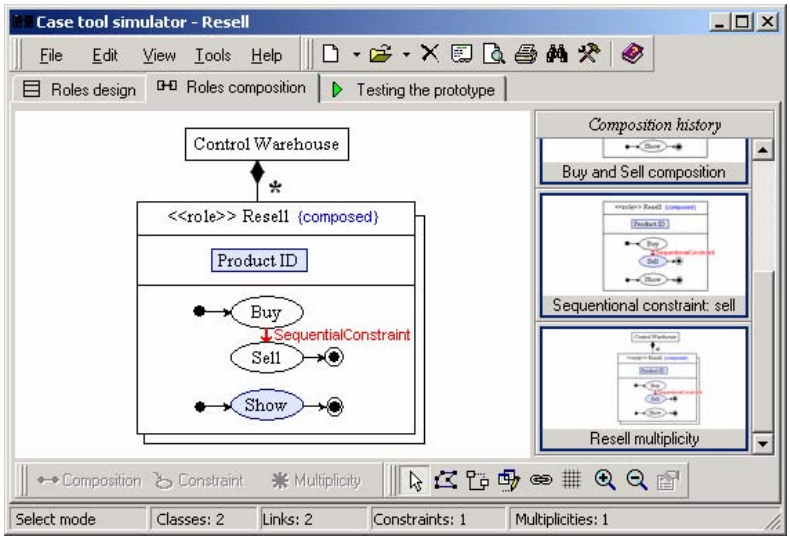


Figure 36. RaP tool: setting up the multiplicity properties

To implement the composition of multiple Reseller roles we provided an additional *ControlWarehouse* class (Figure 36) that implements the Resell role. It stores a list of objects of the *Resell* type. The *ControlWarehouse* class has the same three methods as the Reseller role: *sell*, *buy* and *show*. The method *buy* of the *ControlWarehouse* class creates a new member (the *Resell* role) and then invokes the buy method of this new member. The *sell* method does the following: finds a necessary member (the *Resell* role) in the list of roles, invokes the *sell* method from this role and then deletes this member. The *show* method invokes the corresponding method from each member of the list.

6.4 Summary

In this chapter we have considered the application of our framework in the field of software engineering. We introduced the archetype of the Rapid Prototyping tool that fits our framework. This tool allows for considering a system as a composition of base roles, where each role is small enough to be reasoned about. Base roles are composed into larger roles by documenting explicitly the design decisions in a form of *identity* and *composition constraints* and kept in the history of the design process. This allows system designers to understand how a system is composed from base roles.

To facilitate the analysis of systems composed from multiple roles, we have proposed the RaP tool. This tool is based on the same idea as the method of our framework: prototyping base roles and composing them into a prototype of a system. This method with prototyping capabilities allows a modeler to understand systems and their goals instead of the invasive writing of a supporting code to test models. It helps in the comprehension of models functionality: even people without a deep knowledge of modeling techniques are able to reason about models based on our modeling technique.

Let's see how the RaP tool can be positioned comparing with other research activities in the field of concern-based modeling. We see the following two research directions in this area:

The first direction considers how to model concerns at the code level (so called Technological methods, see our classification of CBDMs in Part 1). AspectJ is the most known concern-based technology and UML is the most known modeling language. Therefore, main research topics here are about modeling the AspectJ code in UML (see [Aldawud01], [Kande02], [Stein02], [Suzuki99] and etc.). Dominik Stein in [Stein02], for example, "… provides representations for all language constructs in AspectJ and specifies an UML implementation of AspectJ's weaving mechanism". These approaches are useful for programmers or software architects. They are not very useful, however, for requirements specifications because they focus mostly on the structure of the AOP code.

The second direction in modeling concerns is related to modeling system requirements without direct link to the code structure, such as [Motschnig99], [Reenskaug96], [Riehle98] (so called conceptual methods, see our classification of CBDMs in Part I). Approaches in this direction make abstractions of implementation details and deal mainly with roles and constraints. The analysis of models in these approaches, however, is difficult. In many cases models can be evaluated only after their implementations.

The RaP tool tries to make a "bridge" between the two upper-mentioned directions. Our framework proposes a concern-based modeling method for requirements engineering and the RaP tool supports it with Java-based rapid prototypes. Even if these prototypes are limited in their functionality and not efficiently implemented, they are useful for system stakeholders to reason about the functionality of the future system.

7 Application to BPM (Conceptual Tool for Business Process Innovations)

In this chapter, we show the application of our framework in the field of business process modeling. We propose to use our framework as the basis for the conceptual tool for business process innovations.

Modern Information and Communication Technology open a door for innovations that can improve the functioning of companies. Many innovations come from the analysis of business processes. Today modeling is widely used for the analysis of business processes. In this chapter we propose a business process modeling technique based on role modeling. To specify a process where one business object may play several roles, a composition operation has to be specified. All role-based techniques have difficulties specifying role composition: composition is never specified without the description of actual messages passing between business roles. Such implementation details complicate the understanding of the model and semantics of composition become implicit. To specify a business process of a complex system at a higher level of abstraction requires the proper understanding of relationships between roles, when they are put together in one common context. In this chapter we use the concept of "composition constraints" defined in our modeling framework. Using "composition constraints" allows a business modeler to make explicit his decisions about how the composition is done. Our approach can be used for building a BPR case tool that enables the discovery of new business processes by means of different disassembling and assembling of roles.

7.1 Introduction

To stay at the competitive edge, the modern market requires companies to adapt new business strategies that allow them to take advantages from modern Information and Communication Technology (ICT). "In the emerging era of the Web, companies that don't have a Web-based strategy - don't have a strategy" [Miers01]. Rapid development of Web-based technologies provides an inexpensive way to communicate with customers, partners and suppliers. A cheap, easy and fast way of the information exchange is the main condition that enables the distribution of activities between different partners. In the "pre-internet era", companies were limited in their communication facilities and therefore the distribution of activities was difficult. Today due to the development of ICT, internal activities of companies often become outsourced to partners and some internal activities may become parts of customer's processes. ICT allows companies to outsource their non-core activities and better define their competence. "… Within the past five to eight years, outsourcing has evolved from a purely tactical option - often of last resort - to an ongoing, standard business practice and strategic management tool" [Berger02].

If we agree that it makes sense to outsource, we have to consider internal activities that can be outsourced. Many outsourcing solutions can be possible. They should be analyzed in two steps. First, a business analyst has to build a model of each solution. Second, solutions should be evaluated and the best one should be taken. In our work we are not going to discuss the second step that determines which outsourcing solution should be taken.

We focus on the question: "How different outsourcing solutions can be found based on the analysis of the business process of a company and existing service providers?" In a company that has a complex business process it can be difficult to identify internal activities that can be outsourced. Internal activities can be so closely interrelated that it can be difficult even to distinguish between them: an internal activity may have many internal constraints (behavioral, structural etc.) with other internal activities. Outsourcing of an activity requires that some internal constraints should be changed in the context of the business process of a company and some internal constraints should be transformed into external constraints (or contracts) between a company and its partners. Also new constraints should be defined between an outsourced activity and activities in the context of the business process of partners. This is a typical Business Process Reengineering (BPR) problem.

Before describing our solution to the mentioned problem we begin with the overview of the business process concept. Business process is only a part of the whole model that shows how the company does its business. Other parts are the business model that have "a description what values the business creates for its stakeholders" [Stähler02] and the implementation that shows how the business process is implemented. Innovations can come from any of these three parts.

In this work, we consider innovations that come from Business Process Part. We take the definition of the business process from the Workflow Reference Model [WMC95]:

Business Process *is "a set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships".*

This definition extends the definition proposed by Davenport & Short: a *business process* is "a set of logically related tasks performed to achieve a defined business outcome" [Davenport90]. It is extended with the notion of roles that are used to specify the responsibilities of objects. *Object* (or business object) is yet another concept that should be considered when the business process has to be implemented. An object is the model of an entity in the Universe of Discourse. It plays roles in a business process by means of participating in different activities. Totally we have to address four key elements: *Goals, Activities*, *Roles* and *Objects*. [Kueng96].

To model these four business process elements some Process Modeling Techniques (PMTs) should be used. Carlsen in [Carlsen97] refers to the following PMTs types: Traditional Input Process Output (IPO) techniques; Conversation Based techniques; PMTs based on role modeling; System thinking and system dynamics techniques; Constrained-based representations techniques. In this chaper, we use PMT based on role modeling. Its main advantage is that it supports the separation of concerns: the best way to deal with the business process of a complex system is to specify concerns separately and show how these concerns are composed together in the context of a business system. Each concern can be specified as a separate activity and a business process is a composition of different concerns.

There are several PMTs based on role modeling. The three of them seem to be the most important: RIN – Role Interaction Networks [Singh92], RAD – Role Activity Diagram [Ould95] and OORAM – the Object-Oriented Role Analysis Method [Reenskaug96]. These three approaches are quite similar. Roles are considered as sets of sequentially ordered actions and/or interactions

The main drawback of the PMTs described above is that goals of business processes are difficult to model with these PMTs. To model goals we have to specify states of business objects because "… goals are usually defined as objectives to be achieved by the system and its environment or a state of affairs that is deemed desirable by stakeholders" [Regev01].

Another important requirement for PMT is that it should be business oriented and thus be simple and diagrammatic. This will allow a business analyst "…to discuss and validate process models with both users and owners of the process, many of whom are not prepared to invest their time in understanding more complex representations" [Phalp97].

PMT that allows for the simulation of a business process can help a business analyst to evaluate a process. Making the simulation possible requires that PMT supports the formalization of a business process. Such rigorous PMT can be used in a BPR case tool for the development and analysis of business processes. We propose a role modeling PMT that satisfies the three mentioned requirements.

As we said above, the role modeling PMT allows for the separation of concerns: a business model is specified as a set of concerns composed together. Therefore, composition is a necessary operation of the role modeling PMT. We show how composition constraints can be used to specify decisions taken by a business analyst in the design process. The specification of composition is done in an implementation independent way that is convenient for the business reasoning.

Role modeling PMT presented in this chapter can be used to build a case tool for the specification and discovery of new business processes by means of different disassembling and reassembling of roles. In this chapter we give two examples. In the first example we show how composition constraints are used to specify a business process as a composition of separate concerns of a system. In the second example we show how reassembling of roles in a business process can help do discover new business processes and models.

7.2 Our Approach for Business Process Modeling

Our modeling language defined in Part II of this thesis can be used to specify all four elements of a business process: roles, activities (or collaborations), goals (postconditions for all roles in a role model) and objects (that play roles in the implementation). Let's give some comments about how business process goals are modeled with postconditions. In our role-based approach, the goal of any collaboration is specified a set of goals for the roles that participate in this collaboration. Roles' goals are specified as a set of postconditions. A business process can include several collaborations (each one represents a separate concern of a system). Therefore, the goal of this business process is a set of goals for each collaboration or the set of postconditions for all roles in a role model.

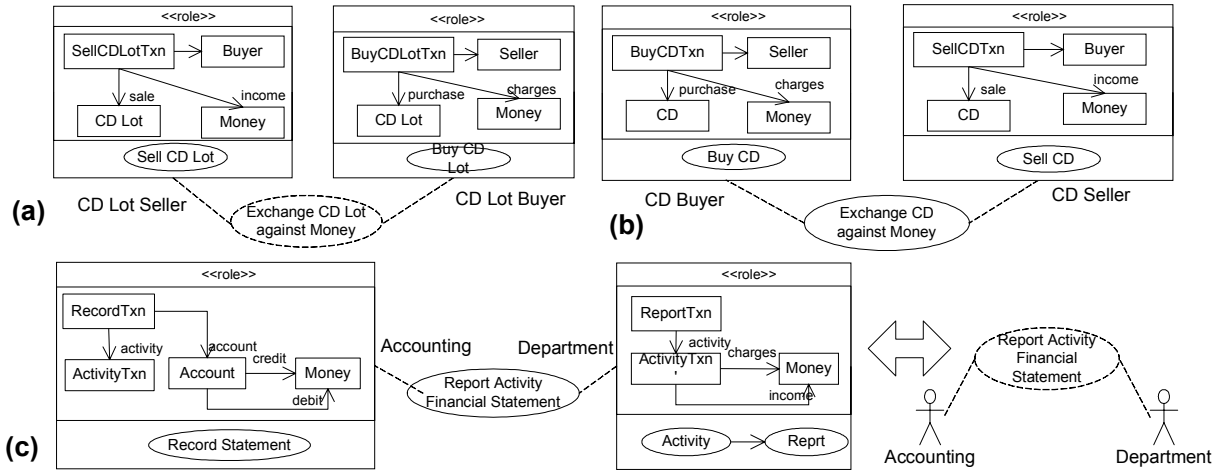


Figure 37. Three base role models a: "Exchange CD Lot against Money", b: "Exchange CD against Money" and c: "Report Activity Financial Statement"

To represent visually the structure of collaborating objects in a business process, we the notation from our modeling framework. We represent a role model (see Figure 37) by collaboration (a dashed oval), roles by stick men or boxes and role names by names below stick men or at the end of a line connecting collaborations with roles. If we want to hide details about roles we represent them with stick men (see Figure 37.c from the right). Otherwise we represent roles by boxes with three panes. We remind you that this notation is similar to the notation used in a UML Collaboration Diagram. The difference is that instead of an attribute compartment in UML (middle pane in each box) we use graphical notation inspired by a UML class diagram to represent a state as a set of attributes and relations between them. Instead of the compartment that holds a list of operations in UML (lower pane in each box) we use the graphical notation inspired by a UML activity diagram to represent a role behavior.

We use this notation to model the three role models that we use as examples in the following sections. They are taken from the StreamCom project supported by the Swiss National Science Foundation (project No. 5003-05755). This project studied issues related to the commercialization of streamed information, such as video, audio and news-feeds. The first two role models (Figure 37.a and 37.b) describe CDs Lot selling/buying activity. The third model (Figure 37.c) specifies how accounting has to be done for a core company department: first the department accomplishes its core business activity and then sends a financial report about this activity to the accounting department. Then the accounting department credits/debits an amount corresponding to the performed activity to/from the company account. In the following sections we show how these role models can be combined to make a composite role model.

Each role model in Figure 37 includes one collaboration and two roles represented by stick men. The goal of each role model is specified as postconditions for each role participating in this activity. Postconditions can be specified formally using the OCL language. In order to simplify the understanding our models without formally specified postconditions, we suggest using names of the role models such that a reader of the diagram can «guess» them. For example, the "Exchange CD Lot against Money" role model assumes the following postconditions: "CDLotBuyer got CD Lot from CDLotSeller" and "CDLotSeller got money from CDLotBuyer".

7.3 Composition of Role in Business Process Modeling

In this section, we show how our PMT based on composition constraints is applied for the specification of business processes. We hope that examples form this section can serve as a prototype for the BPR case tool that can be built based on out PMT.

As we have explained earlier, our role models can be detailed and outlined. A detailed model represents roles as boxes with state and behavioral parts and includes the specification of composition constraints. In subsection 7.3.1 we show an example of how a detailed model can be used to specify a business process form the set of base role models.

An outlined model specifies composition at a higher level of abstraction. It represents roles as stick men and hides details of composition. In subsection 7.3.2 we show an example of how an outlined model can be used in BPR.

7.3.1 Role Model Composition: Detailed Specification

In this subsection we give an example of a detailed specification that shows how a business process can be assembled from the set of base role models.

The idea of this example is the following: based on these three base role models form Figure 37, the goal is to build a new "Retail CD" business process that will specify a CD retailing business. For this we specify the composition of the CDLotBuyer and CDSeller roles in such a way that they share the same account for their business activities. Following this idea, we will compose base roles in two phases (see Figure 38).

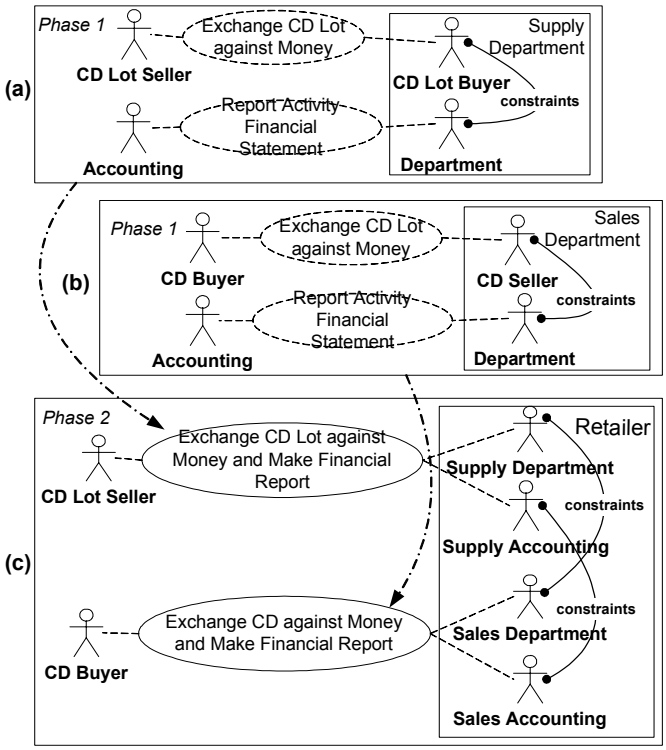


Figure 38. An Example of the composition of role models. (a), (b): Phase 1; (c): Phase2.

In the first phase we include the accounting activity in the "Exchange CD lot against Money" role model: we compose the CDLotBuyer role with the Department role (see Figure 38.a). The result of this composition would be the "Exchange CD Lot against Money and Make Financial Report" role model. In a similar way we include the accounting activity in "Exchange CD against Money" (see Figure 38.b).

In the second phase we compose two resulting role models by means of sharing common accounting activities (see Figure 38.c). We have to specify composition constraints in a way that the Supply Accounting role and the Sales Accounting role share the same account.

Phase 1: The goal of the first phase is to include accounting activities in the "Exchange CD lot against money" and "Exchange CD against money" role models. Two role models would be the results of this phase:

"Exchange CD Lot Against Money and Make Financial Report" =
Compose ("Exchange CD Lot against money"; "Report Activity Financial Statement").

"Exchange CD Against Money and Make Financial Report"
= Compose ("Exchange CD against money"; "Report Activity Financial Statement").

To include accounting activities in the "Exchange CD lot against Money" role model we define the following composition constraints:

Department.A.Activity ⊷ CDLotBuyer.A.BuyCDLot
Department.Attrs.ActivityTxn ⊷ CDLotBuyer.Attrs.BuyCDLotTxn
Department.Attrs.Money ⊷ CDLotBuyer.Attrs.Money
Department.AttrRels.charges ⊷ CDLotBuyer.AttrsRels.charges
Accounting.AttrRels.credit = undef (no credit for buying CD Lot activity)
Department.AttrRels.income = undef (no income for buying CD activity)

These two base role models and implied composition constraints form the role model composition specifications (see Figure 39).

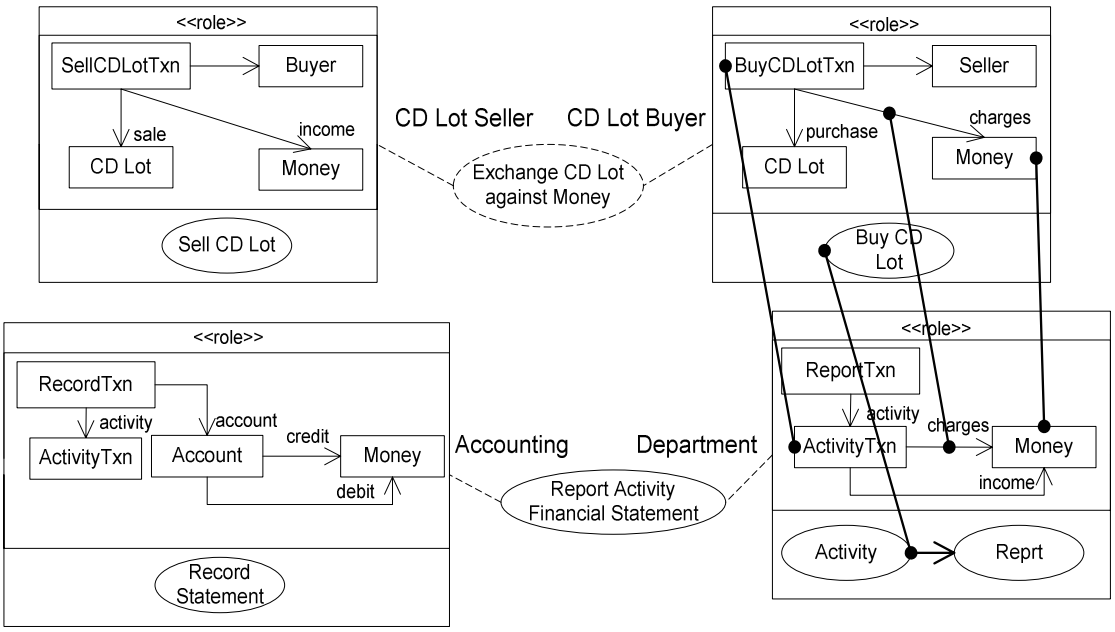


Figure 39. Role model composition specification (phase 1)

The result of the composition of roles from Figure 39 you can see in see Figure 40:

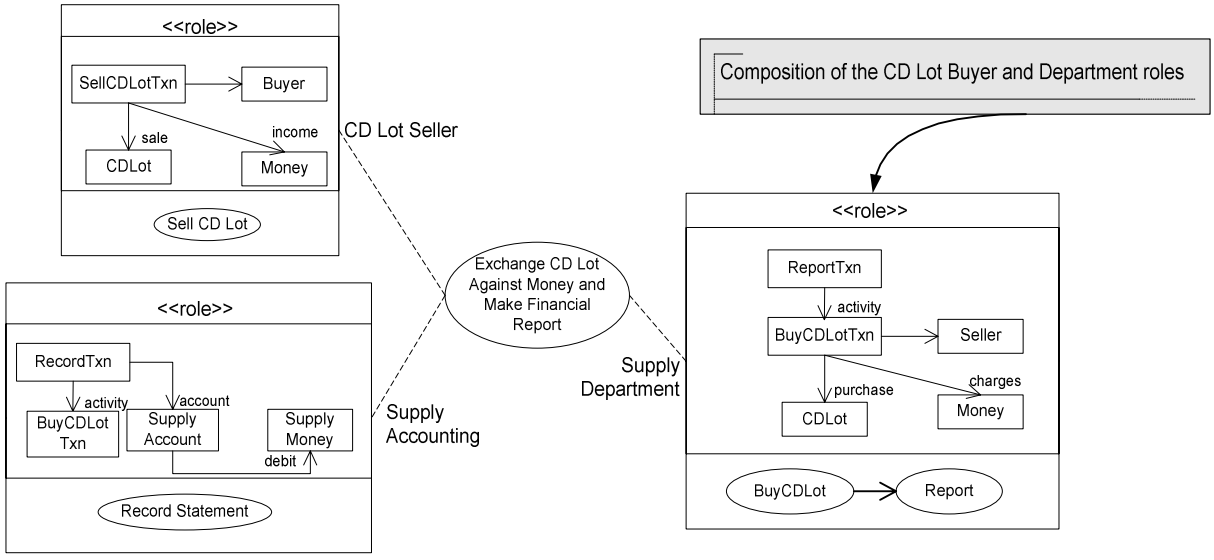


Figure 40. The result of the phase 1

Note, that in the context of the *Supply Accounting* role, we have renamed three attributes: *ActivityTxn* into *BuyCDLotTxn; Money* into *SupplyMoney* and *Account* into *SupplyAccount*. These new names show better semantics of these attributes.

The role model "*Exchange CD against Money and Make Financial Report*", which includes accounting activities in the "*Exchange CD against Money*" role model, can be obtained exactly in the same way as "*Exchange CD Lot against Money and Make Financial Report*".

Phase 2: The goal of the second phase would be to compose two resulting role models by means of sharing common accounting activities. We will obtain the following role model:

"Retail CD" = Compose ("Exchange CD Lot Against Money and Make Financial Report"
"Exchange CD Against Money and Make Financial Report").

In the second phase of role composition, we have to specify composition constraints in a way that the *"Supply Accounting"* role (from the "*Exchange CD Lot against Money and Make Financial Report*" role model) and the *Sales Accounting* role (form the "*Exchange CD against Money and Make Financial Report*" role model) share the same account and money. We also have to guarantee that the Report action (that follows the BuyCDLot action) precedes the SellCD action. Based on this, composition constraints between two role models would be:

$$SupplyAccounting.Attrs.SupplyAccount \bullet\!\!-\!\!\bullet\ SalesAccounting.Attrs.SaleAccount,$$
$$SupplyAccounting.Attrs.SupplyMoney \bullet\!\!-\!\!\bullet\ SalesAccounting.Attrs.SaleMoney$$
$$SupplyDepartment.A.Report \rightarrow SalesDepartment.A.SellCD$$

We show the result of a composition in Figure 41:

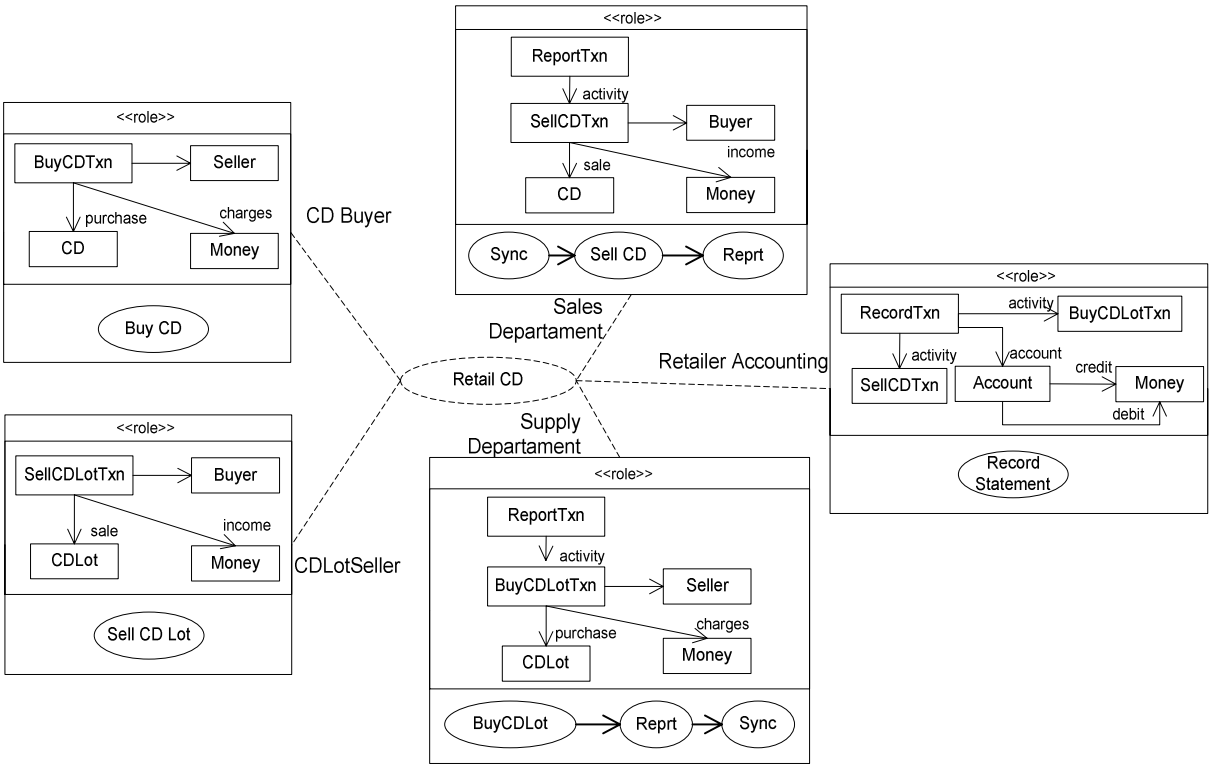


Figure. 41. The "Retail CD" role model (keeping roles corresponding to different departments separately)

We can compose roles a bit further. We can compose the Sales Department, Supply Department and Retailer Accounting roles into one Retailer role. Composition constraints in this case would be that the Money attribute from the Retailer Accounting role should be identical with the Money attribute from the Supply Department and, correspondingly, from the Sales Department. We show the result of the composition in Figure 42. The three roles (Supply Department, Sales Department, R Accounting and Sales Accounting) from Figure 41 become one role: Retailer.

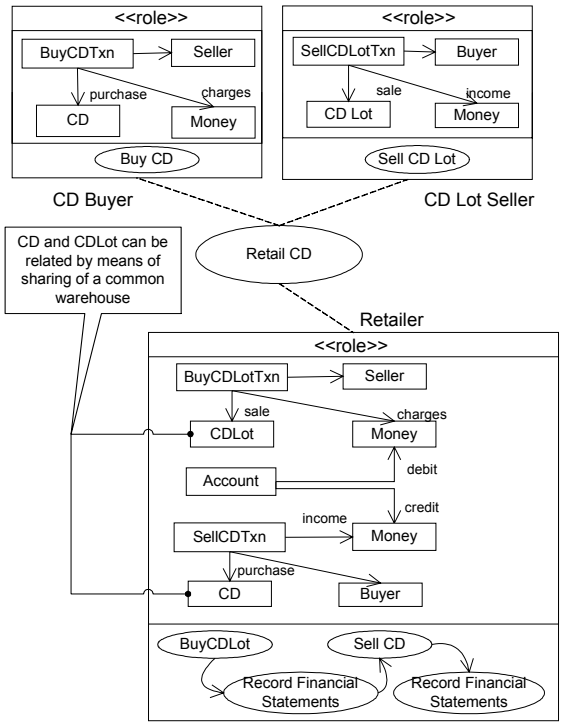


Figure 42. "Retail CD" role model.

Note that in order to define the complete "Retail CD" role model, the "CD Seller" and "CD Buyer" roles have to share not only the common account but also a common warehouse for stocking CDs. The modeling of a common warehouse is done exactly the same way and thus we will not show it.

7.3.2 Role Model Composition: Outlined Specification

In the previous subsection we have considered how a business process can be specified from the set of base role models. The goal of the composition was clearly defined: to specify the "Retail CD" role model. However, we can not always compose role models univocally. Role models can be composed in different ways that can result in different business processes. The important question is: which roles can be composed and how? This decision should be taken by a business analyst based on specifications of the base role models. The experienced business analyst has to rapidly search for meaningful mappings of the model elements in the specifications of roles. Meaningful means that a new composed role model does not contain contradictions and makes sense form the business point of view.

To illustrate how the set of base roles can be used to discover business process models, we have used an example from the audio-streaming industry. In this example we assume that readers are generally familiar with this industry and we do not show detailed specifications of each role. We believe that the names of roles are self-explanatory. The set of base role models was identified by means of analysis of the mp3.com business model (see Figure 43).

The original mp3.com model is shown in the upper left corner of Figure 43. The set of base role models used to specify the mp3.com model is shown in the rectangle in the center of the figure. Around this rectangle we show business processes that can be derived from the set of based role models by means of assigning roles differently to business objects. Here we do not show how roles are composed in the context of business objects. Instead we appeal to our reader's intuition about how composition should be specified with composition constraints.

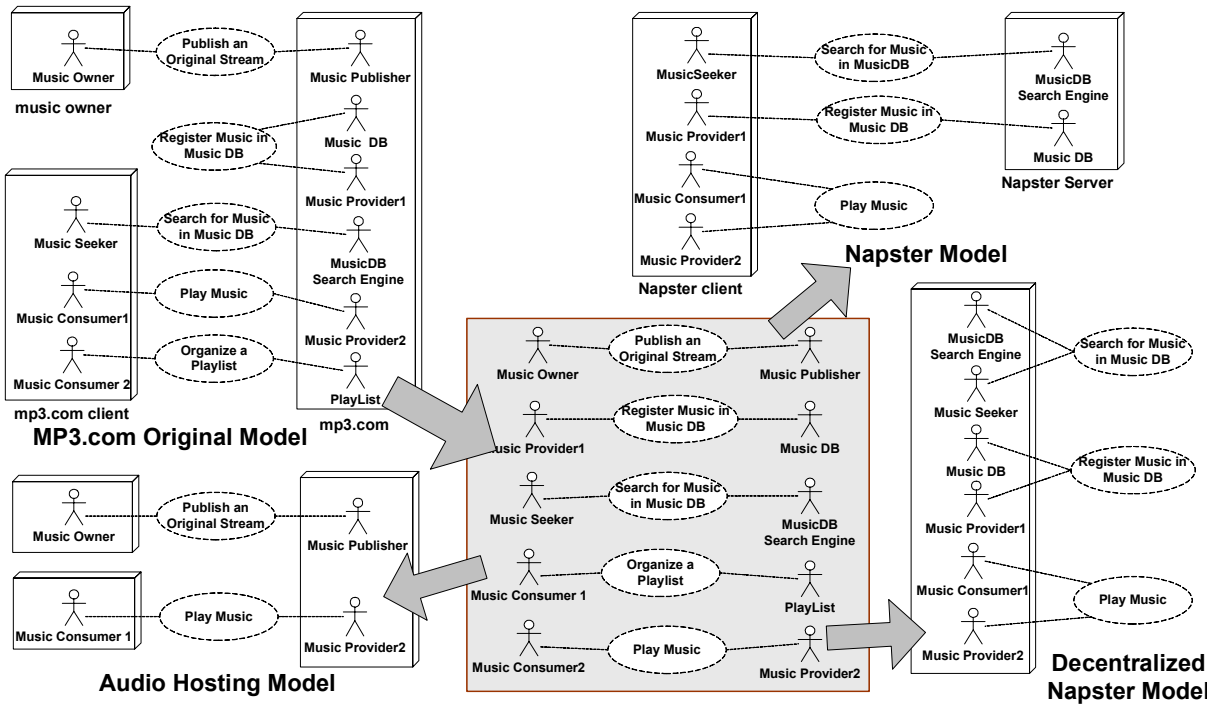


Figure 43. Original MP3.com model and different business processes that can be derived from it.

Role models in Figure 43 can be used as a basis to define new business solutions (like outsourcing solutions). However these role models are not sufficient to define business models. Some additional decisions should be taken by a business analyst. For example, the Napster model from Figure 43 does not have a customer (one that purchases a commodities or services). Thus a business analyst has to define a customer and define how the Napster server earns money. Several solutions are possible. For example: collecting money from Napster clients; making advertisements to Napster clients and collecting money from advertisement's providers. In the actual Napster business model, the Napster server earned money from music owners that were disappointed working with major record companies (such as Columbia or Sony). These music owners sponsored the free distribution of music done by Napster to make their protest against big record companies [Gauze02].

7.4 Summary

In this chapter we applied our framework in the field of business process modeling. We have proposed to use our framework as the basis for the conceptual tool for business process innovations.

We have considered a business process as the composition of different concerns. We used composition constraints to specify how these concerns are grouped together in one business process. To define innovative models we propose to regroup concerns in the existing business process models by means of redefining composition constraints.

Our approach for the specification of business processes based on the composition of role models has strong practical impact. Explicit design decisions on the composition of base roles, specified with composition constraints, allow a business analyst to disassemble and reassemble roles in many different ways and thus create new business solutions. We have considered an example that shows how base roles can be composed in order to create a new business process model. Note that this process can not be automatic. It has to be accompanied by a business analyst who has to make decisions on how the composition of base role models should be done. Our approach can facilitate the work of a business analyst by means of using a business process reengineering case tool that can be built based on role based the process modeling technique proposed in this section.

The idea about restructuring the elements of a business process in order to get a new one (probably better process) is not new. In 1994 Davenport & Stoddard identified seven reengineering myths. One of them is *the Myth of Reengineering Novelty*: "Reengineering, although about familiar concepts, is new in that these concepts are combined in a new composition. These key components have never been together before" [Malhotra98]. This shows that composition of independent concepts is one the most important operations in BPR. Different models of composition have been proposed by [Riehle98], [Fiadeiro01], [Reenskaug96], [Bernstein99]. Some of them consider composition at a very detailed level where the semantics of composition becomes hidden behind technical details (such as message passing); others do not consider the explicit modeling of state that makes the goal modeling impossible.

In our approach we propose composition constraints to specify composition independently from implementation and allow for mapping structural and behavioral model elements. We believe that these two main features of our method can accelerate the work of a business analyst and improve the comprehensibility of business process models.

8 Application to BPM (Capturing Design Rationale with Roles in Business Processes Modeling)

In this chapter, we show one more application of our framework in the field of business process modeling. We propose to use the modeling method from our framework to capture the design rationale in business process modeling.

The permanent improvements in IT technologies make available new business support systems (BSSs). These BSSs allow for a better integration between business processes and BSS systems. When a new technology becomes avail-able, the existing business process has to be reconsidered for the automation with a new BSS. This reconsideration of the existing business processes can be made easier if the documentation of the business process design rationale is made explicit. Making design rationale explicit allows for better understanding of business processes and, therefore, allows for faster adaptations of the existing processes to new BSSs. The most common way to represent design ration-ale is to enable the traceability from business process model elements to design rationale arguments. In our work we do not go into details about the kinds of arguments that should be used: all of them can be used de-pending on the situation. We concentrate on a specific modeling process that makes capturing design rationale straightforward. Putting the accent on the modeling process rather on the design rationale arguments distinguishes our work from other works in this field. Our modeling process consists in the specification of small parts (roles) of a business process that are composed together into the eventual business process model. In our method we encourage a designer to think about the base roles involved in a business process and define explicitly how these roles are composed together. This facilitates business process re-engineering and improves the alignment between business processes and BSSs.

8.1 Introduction

In many companies, especially in manufacturing, the core of business processes does not change much over years. The same process can be used for several years while being slowly adapted to new technologies. On one hand, the life cycle of such business processes is long: the conceptualization, construction and operation may take several years. During that time original business process stakeholders (designers and participants) may be replaced with new stakeholders. The result of changing stakeholders is that the *design rationale* that explains why a business process is done in a certain way may become lost. This happens because the documentation of the design rationale is often considered a routine, is not made explicit and, therefore, becomes lost with changing business process stakeholders.

On the other hand, the life cycle of business support systems (BSS) is short. The improvements in IT technologies permanently provide new BSSs that allow for better integration between business processes and BSSs. When a new technology becomes available, the existing business process has to be reconsidered for the automation with a new BSS. This re-examination of existing business processes can be made easier if the documentation of the design rationale for these business processes is made explicit.

In large companies, business processes are complex. Models of complex business processes are difficult to understand without a clear picture of a *design rationale* that justifies the existing design commitments. A business process designer has to understand *why* the process is the way it is and *how* the existing business process commitments were reached. If the designer asks the employees that participate in the business process why certain design decisions were made, the most common answer would be that these design decisions have proved themselves over time: the existing business process had worked successfully for many years. In many cases the traces that explain the design rationale are lost and finding the person who made a certain design decision is not easy.

The lack of rationale in business process models results in incorrect solutions: some parts of the business process may be missing or misinterpreted. The solution to this problem is to make explicit how business processes are designed. “A minimum requirement for a design knowledge management capability is that it must be able to record historical precedence, as well as statements of beliefs and rationalizations for why a current design situation is identical to a previous one” [Mayer95].

However, the documentation of the design rationale is considered a routine. “It is unreasonable to expect designers to introduce a separate step in the design process to document, or model, the assumptions or rationale upon which a given design process is based. Therefore, much of the capture of this information must occur through background processes or interactive questioning initiated by a design support environment, rather than the designer” [Mayer92]. In our work we propose a specific method where this design rationale background process consists in the specification of small parts (roles) of a business process that are composed together into the eventual business process model. In our method we encourage the designer to think about the base roles

involved in a business process and define explicitly how these roles are composed together. This improves the understandability of business process models, increases the reusability of model elements and facilitates business process re-engineering.

Our method is based on role modeling. Why are roles convenient for business process modeling? The rationale behind role modeling is the following: a business process model usually mixes the knowledge that belongs to different participants of the business process; it represents a synthetic view of a super-observer that observes interactions of different participants. Frequently, why the business process is modeled in a certain way is explained with goals. However, the overall goal of the business process can be difficult to understand, because the business process may include many different roles played by different participants with their own goals. Therefore, we propose to represent the business process as a composition of roles. This helps a modeler to understand the objectives (they can be modeled with Actor Dependency diagrams, see [Yu94]) of each participant and helps to explain the design rationale behind the business process. Furthermore, modeling with roles makes it easier to modify business process models when changes happen in roles as the results of regulatory, technical, or social changes in the business environment of a company.

We remind your that in our modeling approach we represent a business process as the collaboration (a dashed oval) of roles (see Figure 44). We understand roles to be the crafts of business process participants (or stakeholders), i.e. the knowledge of the specialists that are involved in the business process. We represent a role with a rectangle that includes a set of actions, sequential constraints between them, and the tools and materials that a specialist needs in his craft to perform the actions. We represent physical objects with cubes. Physical objects can be: a company, facilities, tools or materials.

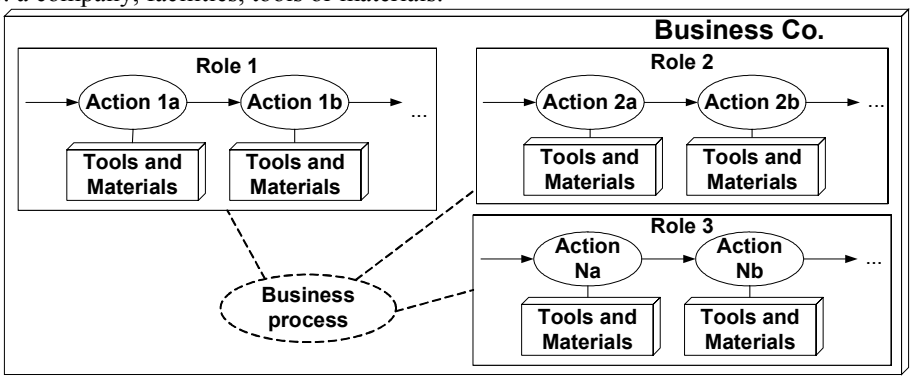


Figure 44. Business process representation in our approach

We believe that our approach improves the understandability of business process models with the explicit modeling of the design rationale for business processes. Thinking in terms of roles helps a modeler to understand the crafts of the business process stakeholders and to see how these crafts are used in the business process.

To illustrate the second application of our framework in the field of business process modeling, we use the example of a manufacturing process in Pharma Co., a pharmaceutical company. This company needed to introduce a MES (Manufacturing Execution System) to ensure the order of the manufacturing process. The goal of the "MES" project was to formalize the manufacturing process that will be controlled by MES. In this project our method was used to ensure the correct understanding of the manufacturing process by a modeler and to ensure that the model of the manufacturing process correctly reflects the viewpoint of all participants in this process.

In Section 2 we explain how our work differs from other works that allows for capturing design rationale. In section 3 we explain our method with an example of the Pharma Co. First, in Section 3.1 we give an overview of the manufacturing process of Pharma Co. Then in Section 3.2 we present base roles of this process and explain the composition of these roles in section 3.3. Section 4 is the conclusion.

8.2 Capturing Design Rationale: Overview

Capturing design rationale has been a research topic for a long time starting from the early 1980s. A comprehensive introduction to design rationale you can find in "The Encyclopedia of Computer Science and Technology" [Buckingham96]. We refer to [Mayer95] for the definition of design rationale: "The beliefs and facts as well as their organization that the human uses to make (or justify) design commitments and to propagate those commitments". The most common way to represent design rationale is to enable the traceability (usually in the semi-formal way) from project artifacts to design rationale arguments.
According to [Mayer95], the design rationale arguments can be:

- The philosophy of a design that describes on the high level of abstraction the behavior and structure of a system of interest;

- Design limitation expressed as range restrictions on systems parameters, and environment factors;
- Factors considered in tradeoff decisions, including budget constraints, timing constraints, organization policies and procedures, and availability of technology;
- Design goals (including components that have to be used, priorities on problems requirements,)
- Precedence of historical proof of viability:
- Legislative, social, professional, business, or personal evaluation factors or constraints.

In our work we do not go into details about the kinds of arguments that should be used: all of them can be used depending on the situation. We concentrate on a specific process that allows for capturing design rationale. Putting the accent on the process rather on the design rationale arguments distinguishes our work from other works in this field.

Among approaches that support the design rationale argumentation, the design rationale is usually captured in the form of the design history (see the following examples: Design Space Analysis (DSA) with the Design QOC notation, proposed by MacLean et al. [MacLean91]; the Issue Based Information System (IBIS) notation and tool [Conklin89], [Conklin91]; Decision Representation Language (DRL) [Lee91] with the corresponding tool called SIBYL [Lee90]). The design history is represented as the network of intermediate artifacts that have to be reached in order to get the eventual model. Each artifact is modeled with its issues or (goals) that have to be solved. Several alternatives are considered as means to solve the mentioned issues. Alternatives are justified with the design rationale arguments (justifications). One alternative has to be taken by means of considering different arguments. Once the alternative is taken, new artifacts have to be developed for this alternative. Then the process repeats: new issues are identified, the corresponding alternatives are considered for these issues, one alternative is taken and new artifacts are developed.

In the context of business process modeling we have to mention the Design Rationale Capture Method IDEFF6 [Mayer95]. This method makes explicit the traces from IDEF model elements to the aforementioned design rationale arguments. The main accent in this method is given to the types of the design rationale arguments that can be used. However, IDEF6 does not explain how the design rationale history should be built. In IDEF the design process consists of several refinement steps, where each step is done in ad-hoc way. Therefore, it may be quite difficult to understand the design rationale behind the refinement of IDEF models. In addition, the design rationale for each refinement step may include the arguments of very different business process stakeholders, which also complicates understanding. Therefore our goal was to bring in a process that makes the design rationale more intuitive and that keeps separately the design rationale for different business process stakeholders.

In our work we propose a method based on role modeling where roles represent the crafts of business process stakeholders. There are several process modeling techniques based on role modeling. Three of them seem to be the most important: RIN – Role Interaction Networks [Singh92], RAD – Role Activity Diagram [Ould95] and OORAM – the Object-Oriented Role Analysis Method [Reenskaug96]. These three techniques are quite similar. Roles are considered as sets of sequentially ordered actions and/or interactions.
Our approach differs from the aforementioned role modeling techniques by the following characteristics:

- We define roles as crafts of business process stakeholders. In the aforementioned role-based modeling techniques roles are defined by means of identifying the behavior of the direct business process participants. However, these direct participants constitute only a part of business process stakeholders. We believe that defining roles related to other stakeholders (such as process engineers: logistic engineer, quality assurance engineer, process planner) can improve the model clarity. In this section we consider roles as the artifacts of manufacturing process engineers that design different aspects of a manufacturing process. This allows for better understanding each role and how all roles are composed into the eventual business process model.
- In our approach we represent physical objects, attributes or concepts that a role needs to execute its actions [Wegmann03], [Balabko03a]. The other role modeling techniques represent roles the sequence of actions, keeping physical objects and attributes implicit.
- We represent explicitly the composition constraints that allow a modeler to keep traces of how the eventual business process model was constructed from the set of base roles. Our composition constraints allow for constraining not only the sequence of actions between base roles (as in other aforementioned methods) but also the composition relations between attributes or concepts of base roles.

In the conclusion of this section we want to emphasize that in our method we concentrate on a modeling process rather than on the design rationale arguments that are used during this process. Our method encourages designers to think about roles of manufacturing process stakeholders and defines explicitly how these roles are composed together.

8.3 Pharma Co. Manufacturing Process Modeling with Roles

In this section we explain how we used our method for modeling Pharma's manufacturing process. This process consists of two parts. First, medicine is manufactured in bulks. Second, these bulks are filled into bottles or tubes and then packed. We take an example from the first part of a manufacturing process.

8.3.1 Overview of our Method Capturing Design Rationale

Based on our method, we specify a manufacturing process as the hierarchical composition of roles. On the lowest level of the hierarchy, roles (we call them base roles) represent the artifacts of manufacturing process engineers: each base role is the partial model of a manufacturing process specific to the matter of a certain engineers. In Figure 45 you can see the following base roles: the product manufacturing, raw material provider, product sender, Quality Control (QC), cleaning and cell maintenance roles. These base roles are gradually composed into the eventual manufacturing process model.
At each level of the hierarchy the composition is carried out in two steps:

- Step one: Specify base roles that have to be composed or merged into the eventual model. Base roles are specified by (or in a collaboration with) corresponding engineers. For example, the QC role is designed by a quality assurance engineer. The design rationale for each base role is a matter of the corresponding engineer and, therefore, in our process we do not explain why based roles are designed in a certain way.
- Step two: Base roles are composed or merged into the eventual model. At this step there may be several alternatives for the composition of roles (for example, different sequences of actions in the role composition can be defined). Each alternative is specified with *composition constraints.* Composition constraints specify the relation of dependance between roles that have to be composed. One of these alternatives is taken based on the design rationale arguments (such as arguments mentioned in Section 2). Then the composition of roles is carried out which results in a new composed role (new partial model of a manufacturing process).

These two steps are repeated until the eventual manufacturing model is achieved. Capturing design rationale with our method is quite straightforward: a designer has to capture the artifacts (or roles) of business process engineers and the composition constraints that capture how these roles are composed together.

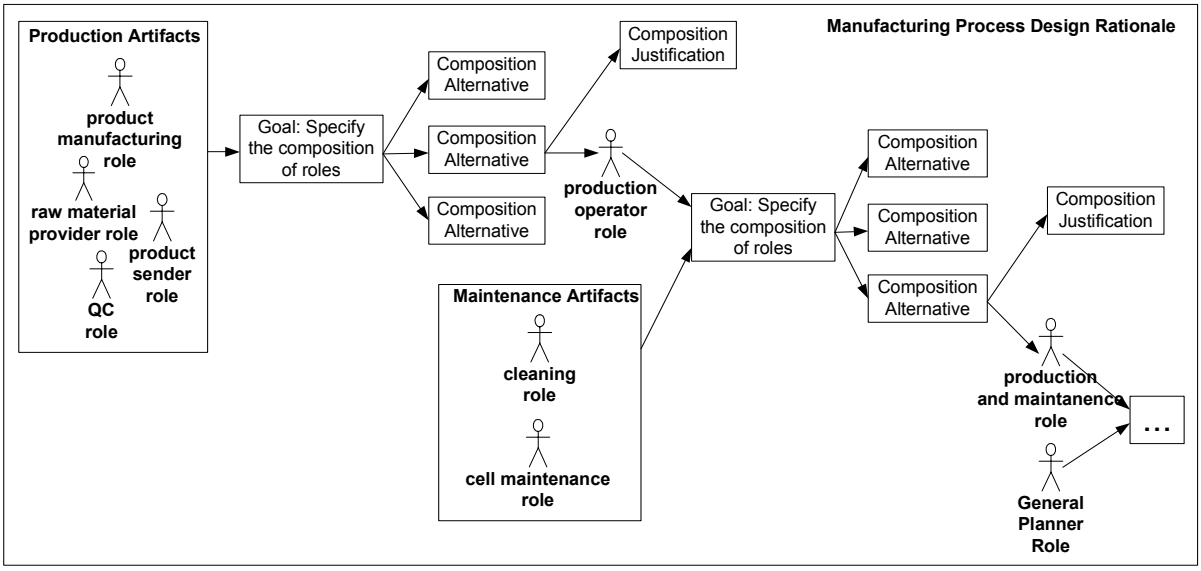


Figure 45. The model of a manufacturing process based on the composition of base roles

8.3.2 Example of a Manufacturing Process

In the example that we use in our work we show only the part of the process that you can see in Figure 46: we show how a model of the "production operator role" is specified as the composition of base roles (roles in the "Production Artifacts" box in Figure 45). In this subsection we begin with an overview of the "production operator role" model (see Figure 46). This model is used by the operator of the manufacturing process who is responsible for the execution of all actions in this process. This model is based on the internal notation developed

in Pharma Co. and inspired by IDEF [NIST93] and UML Activity Diagrams. It consists of blocks representing actions of a manufacturing process.

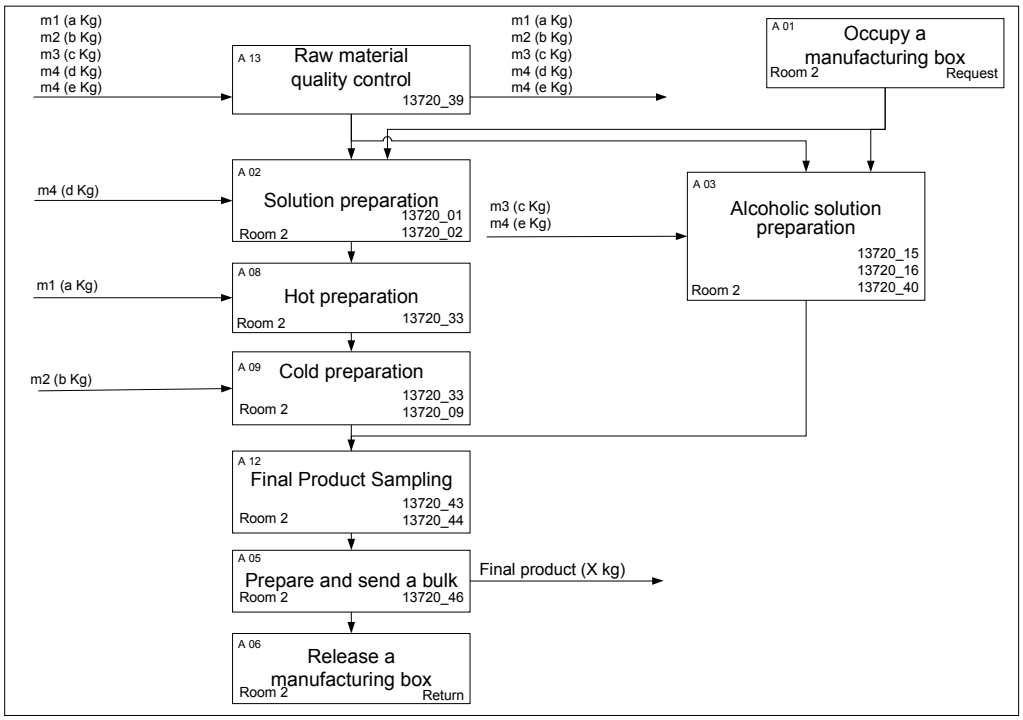


Figure 46. The model of the manufacturing process in Pharma Co

Each block contains the following information about the corresponding action:

- Action name and description (in the center of the block);
- Identifier of the action (upper-left corner);
- A manufacturing room in which the action is executed (lower-left corner);
- Identifiers of sub-processes: these identifiers link each block to more detailed sub-processes specified in a similar way.

Blocks have also incoming and outgoing arrows that represent:

- Horizontal incoming: consumed products (names and quantities of products are given above arrows);
- Horizontal outgoing: created products (names and quantities of products are given above arrows);
- Vertical incoming and outgoing: sequential constraints between actions that specify the sequence of actions. Note that actions can be executed in parallel, for example, the "Solution preparation" and "Alcoholic solution preparation" actions are executed in parallel after the "Raw material control" action.

Based on this notation we can see that the operator of the manufacturing process has to occupy a manufacturing room, receive raw materials, complete the quality control of these materials and then manufacture a final product. To finalize the manufacturing process, the operator has to complete the quality control by sampling the product (then samples have to be analyzed by the quality control department).

In the following section we explain how this manufacturing process can be specified as the composition of roles (we call them "base roles"), where each base role represents a view of a certain engineer of the Pharma Co. company.

8.3.3 Base Roles of a Manufacturing Process

The manufacturing process from the previous section can be decomposed in four base roles: raw material provider, product manufacturing, bulk preparation and QC roles (see Figure 47). These four roles are performed by a manufacturing operator who manufactures a final product from raw materials using tools from a manufacturing floor.

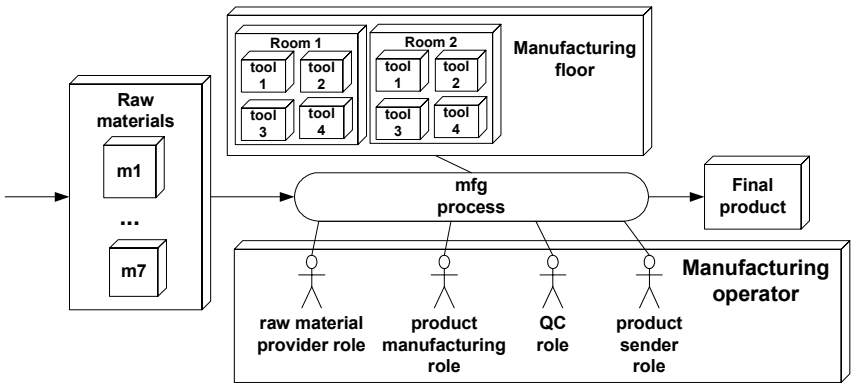


Figure 47. Manufacturing process

In the following sections we present each role and explain how the composition of these roles can be done.

Product Manufacturing Role. The Product Manufacturing Role is the core role of the manufacturing process that specifies how a Final Product is created from raw materials (see Figure 48). In this role four actions consume raw materials. We indicate this by changing the multiplicity of raw materials (mX objects) from one to zero (1→0) and with arrows that comes from consumed materials to the corresponding actions. The "Mix" action results in the creation of the Final Product. We indicate this by changing the multiplicity of the Final Product from zero to one (0→1) and with the arrow that comes from this action to the Final Product. The model from Figure 48 also specifies tools that are used in the manufacturing process. For example, Tool 2 is used in the "Alcoholic solution preparation". Tool 1 is used in all other actions.

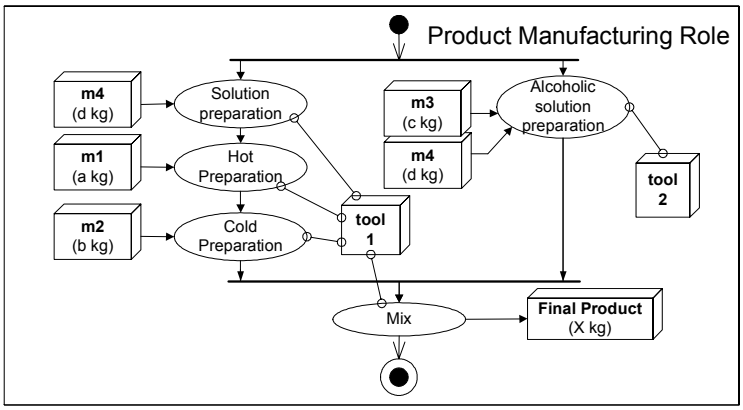


Figure 48. Product Manufacturing Role

Process Interface Roles. The *raw material provider* role defines the action of receiving raw materials from a warehouse. The result of this action is that multiple raw materials become available. We indicate this in Figure 49.a by changing the multiplicity of materials (m*X*) from zero to multiple (0→*) and with the arrow that comes from the "mX" object to the "Get raw materials" action. The *product sender* role defines the "Prepare and send a final product" action. We indicate that the product was send by changing its multiplicity (1→0).

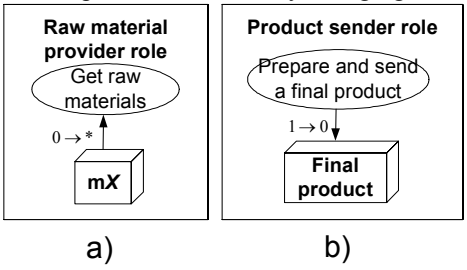


Fig. 49. Roles of a Manufacturing Process: a) Raw material provider; b) Product sender role

Quality Control Role. The *QC* (quality control) *role* specifies two quality control actions. First one is used for the control of raw materials before the manufacturing process (raw materials change state from unchecked to

checked, see Figure 50). Second one is used for the control of the manufactured product after the manufacturing process (sampling the manufactured product).

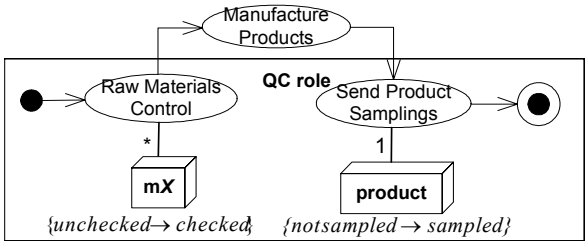


Figure 50. Quality control roles

Manufacturing Floor Role. The main idea of the manufacturing floor role (see Figure 51) is to specify states of the manufacturing floor rooms. The manufacturing floor role changes its state from "clean" to "occupied" in the Occupy action and from "occupied" to "dirty" in the Release action.

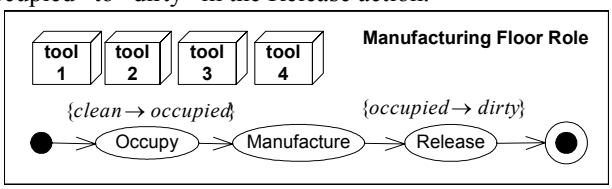


Figure 51. Manufacturing floor role

8.3.4 Composition of Base Roles

In this section we explain how the above mentioned roles are composed together. The composition is done by means of *composition constraints:* constraints implied on the behavior of roles to be composed. Composition constraints show how the composed roles are related to each other. In this section we use two types of composition constratins:

- **Constraints of sequentiality:** define the sequence of actions of roles to be composed. For example, in Figure 52 the "Get raw materials" action (Raw material provider role) precedes the "Raw materials control" action (QC role).
- **Identity constraints**: two model elements are identical if they represent the same entity in the reality perceived by modelers. For example, raw materials received in the "Get raw material action" are identical with materials to be checked by the "Raw material control" action.

Note that in Figure 52 we show all constraints of sequentiality (we represent them with dotted arrows between constrained actions) and only some identity constraints related mainly to the QC role (we represent them with dashed lines between identical objects).

The business process model specified with roles from Figure 52 is similar to the process from Figure 46 with the difference that tools (Tool1 and Tool2) are not shown explicitly in Figure 46. In Figure 46 these tools are specified in sub-processes of the main process.

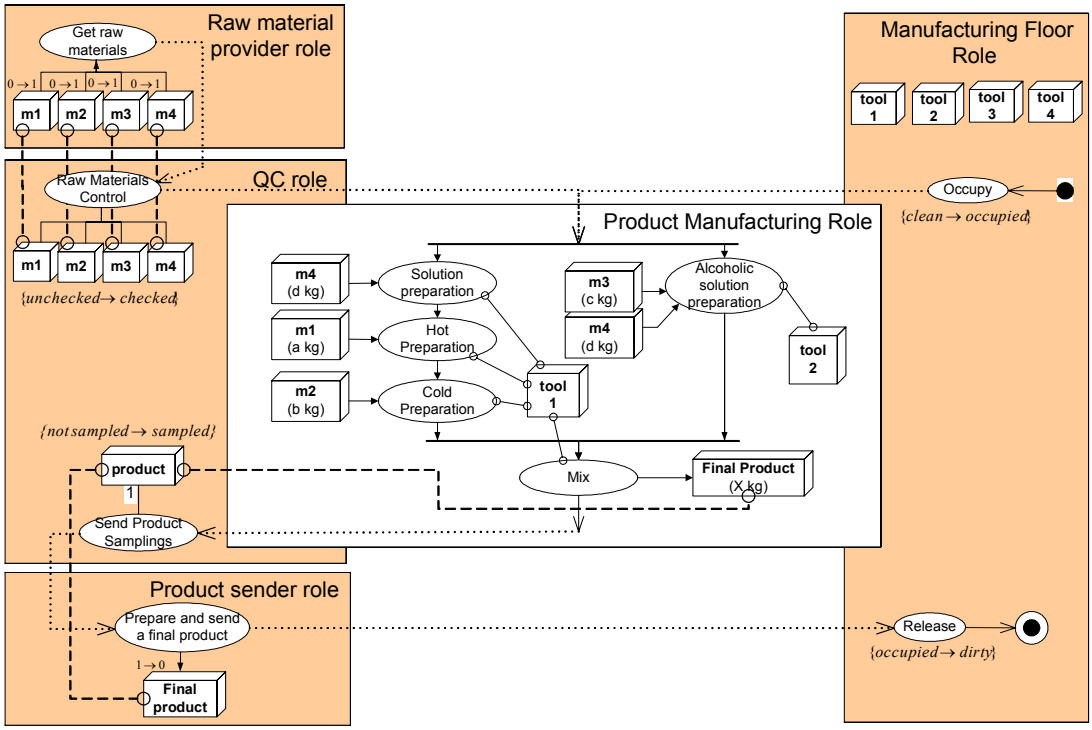


Figure 52. Manufacturing process as a composition of base roles

Up to this point we have explained how a manufacturing process can be specified as the composition of multiple roles. Let's see why we decided to take the four aforementioned roles. The main reason for this is that each role represents a view of a certain engineer involved in the design of the manufacturing process (see Figure 53):

- *Product manufacturing role* is the view of a *manufacturing process engineer*: Its specification is based on the formulas provided by pharmaceutical chemists and the available manufacturing tools.
- *Raw material provider* and *product sender roles* are the views of a *logistic engineer*. These roles specify the distribution of raw materials and manufactured products.
- *Manufacturing tools* are the views of a *manufacturing tools engineer*: specifies manufacturing tools to be used in the manufacturing process.
- *QC role* is the view of a *QA (quality assurance) engineer*: This role specifies quality control actions in the manufacturing process.

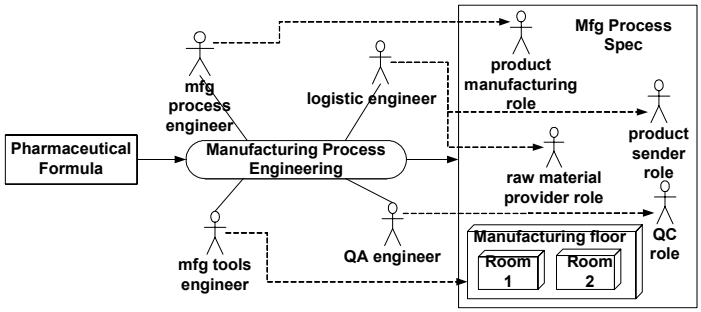


Figure 53. Manufacturing process engineering and its artifacts

Our model of the manufacturing process makes explicit the views of different engineers and the way these views are composed into one model. We believe that defining roles that represent the crafts of manufacturing engineers improves the clarity a manufacturing process model: this allows for better understanding how the roles (or artifacts) developed by the engineers and how these roles are composed into the eventual manufacturing process model.

8.4 Summary

In this section, we adapted the modeling method from our framework to capture the design rationale in business process modeling. Our method enables a designer to make explicit the way a business process model is built.

Capturing the explicit design decisions allows for a better re-examination of the existing business process models in order to adapt them to new business support systems. Our method based on the composition of roles improves the overall understanding of the process: it becomes clear why the process is specified in a certain way. Every role can be separately discussed with a specific stakeholder (engineer). The design decisions about the composition of these roles are made explicit. This makes it easier to reengineer a process in a changing business environment: if a certain role of a process has to be changed as a result of a regulatory, technical, or social change then it becomes easier to see how this change will be reflected in the whole process. For example, the QC role is very sensitive to regulatory and technological changes. If the FDA (American Food and Drug Administration) changes its regulation and requires some verification to be made during the manufacturing process, the process will need to be changed to maintain the fit with new FDA regulations. Similarly, if a technological change enables the company to perform an automatic quality control where the control is manual today, the process has to reflect this change. Furthermore, the QA engineer must always check for new technologies that can improve the QC. The QC role can be used to specify what needs to be checked to maintain the quality control actions (regulatory or technical changes). This example shows that our approach can be used to maintain the fit between the business process and its environment by means of tracking changes in the environment on a per role basis.

The drawback of our method is that it requires a lot of "paper" work without an appropriate case-tool. The diagram that represents the result of the composition (Figure 46) should be automatically generated from the diagram with base roles and composition constraints (Figure 52). However, without the tool this work should be done manually. This takes time, introduces "copy/paste" mistakes and inconsistencies.

Part III Summary

In this section we have described three applications of our framework in software engineering and business process modeling domains. In the domain of software engineering:

- We used our framework as the basis for the Rapid Prototyping (RaP) tool based on Aspect-Oriented and Subject-Oriented programming. We introduced the prototype of the RaP tool that fits our framework. This tool allows for considering a system as a composition of base roles, where each role is small enough to be reasoned about. Base roles are composed into larger roles by documenting explicitly the design decisions in a form of *identity* and *composition constraints* and kept in the history of the design process. This allows system designers to understand how a system is composed from base roles.

In the domain of business process modeling:

- We have proposed to use our framework as the basis for the conceptual tool for business process innovations. In our approach a business process is defined as the composition of different concerns. We used composition constraints to specify how these concerns are grouped together in one business process model. To define new, innovative models we have proposed to regroup concerns in the existing models by means of redefining composition constraints.
- We adapted the modeling method from our framework to capture the design rationale in business process modeling. In particular, we have used our framework to specify and capture the design rationale in a manufacturing process

All three applications can be considered as practical validations of our framework, especially, the last one. It was done in a collaborative project with a big pharmaceutical company. This company needed to introduce a MES (Manufacturing Execution System) to ensure the order of the manufacturing process. The project members acknowledged the ability of our framework to improve the comprehensibility of a manufacturing process.

CONCLUSION

Main Results

In this thesis we have defined the Situation-Based Modeling Framework in the context of Enterprise Architecture (EA). This framework improves system modeling by making models more systemic and, therefore, making reasoning about these models easier.

This work was inspired by the analysis of existing Concern-Based Design Methods (CBDMs) in the context of EA. In this analysis we have studied the ontological, epistemological, theoretical and methodological principles of the CBDMs. The result of the analysis shows that the support of epistemological principles in the existing CBDMs is often missing. Without epistemological principles, CBDMs result in models that difficult to understand and reasoning about.

Our Situation-Based Modeling Framework solves the aforementioned problem. The novelty of our framework is its foundation in epistemological principles that we have identified by analyzing the existing CBDMs:

- Situation Relativity principle: a modeling method should support the explicit modeling of situations;
- "Goal Driven" principle: a modeling method should use goals to represent results to be achieved by a system, placed in a given situation;
- State and Behavior Holism principle: a modeling method should not tend to separate the state and behavior models;
- Diagrammatic representation principle: Models built in the context of EA should be diagrammatic;

In order to make situations the first-class citizens in our modeling framework, we took the key principles for our framework from the situation theory as the foundation for the ontology and the theory of our framework:

- Situation-based modeling: Any entity (a system) should be considered in a number of situations. Any assertion about system properties depends on a situation.
- Explicit modeling of constraints: the mutual relations between system roles should be modeled explicitly.
- Hierarchical modeling: The behavior of an enterprise system should be represented in the form of the role hierarchy that corresponds to the hierarchy of situations.

In our framework a modeler have to make explicit different situations where the system of interest should be defined. Each model element should be defined in a relation to one (or several) of these situations. This situation-based approach makes the definition of model elements more precise. As a result, the whole model defined as the composition of models' parts (related to the different situations) becomes more rigorous and thus understandable.

We position our Situation-Based Modeling Framework as a generic one in the context of EA. This means that our framework can be used for modeling at different organization levels, from marketing to IT. Our framework does not prescribe any specific set of enterprise architecture deliverables. Instead it can be used as a basis to define specific artifacts that should be created at any given level of EA hierarchy. Depending on the organization level (such as marketing, business process or IT levels) our framework can be adapted for the specific needs: a modeler can define specific artifacts and define a specific method.

Impact

The main impact of this thesis consists in making systems' functional models more systemic. Such models are more precise, understandable and human friendly. We have confirmed the impact of our modeling framework with three applications.

In the domain of software engineering:

- We used our framework as the basis for the Rapid Prototyping Tool based on Aspect-Oriented and Subject-Oriented programming. The goal of this tool is to facilitate the analysis of systems specified as the composition of multiple roles. This tool is based on the same idea as the method of our framework: prototyping base roles and composing them into a prototype of a system. It allows a modeler to concentrate on the design decisions instead of the invasive writing of a supporting code to test models.

In the domain of business process modeling we applied our framework for two applications:

- We have proposed to use our framework as the basis for the conceptual tool for business process innovations. In our approach a business process is defined as the composition of different roles. We used composition constraints to specify how these roles are grouped together in one business process model. To define new, innovative models we have proposed to regroup roles in the existing models by means of redefining existing composition constraints.

- We have proposed our framework to capture the design rationale in modeling manufacturing processes. Our framework provides an elegant way to make roles explicit in specifications of manufacturing processes and to make explicit the design rationale about the composition of these roles. This application was done in collaboration with an industrial partner and served as a practical validation of our framework.

Another practical impact of this thesis is a step forward in categorizing numerous CBDMs that can be used in the context of EA. The separation of concerns in system analysis and design has already quite a long history. First methods that used the separation of concerns (such as the early version of the Object Role Modeling method [Mark87], [Halpin01] and Role Activity Diagrams [Holt83]) appeared in the seventies and the early nineties of the twenties century. Since that time a plenty of CBDMs were developed. Many of them can be used in the context of EA. The huge number of CBDMs and their diversity make it difficult to use CBDMs in EA in a coherent way. This variety of methods can cause two problems for those who develop and use innovative CBDMs in the field of EA. The first problem is to choose specific CBDMs that can be used in a given EA methodology: this is a problem for researchers who develop their own EA methodology. The second problem is to find similar methods (with the same problematic or with similar frameworks) in order to make a comparative analysis with these methods: this is a problem of researchers who develop their own CBDM related to a specific problematic in EA (such as business process modeling or aspect oriented programming). To be able to answer to both these questions, we have defined the set of requirements for concern-based analysis and design. Based on these requirements we have made the classification of the existing CBDMs. These requirements and the classification of CBDMs can help researchers and practitioners to become orientated in the mass of the existing CBDMs. In particular, they can be used:

- To evaluate, compare and choose appropriate CBDMs for using them in EA projects;
- To develop new CBDMs in the context of EA or to extend existing methods with concern-based analysis and design.

We finish the list of impacts with the feature that emerges from our generic framework and the classification of the existing CBDMs that we have provided in our work:

- Our framework (including the state of the art section) can be used as the integration tool in the context of EA. Usually EA models will include several CBDMs (such RAD at business process organization level and Aspect-Oriented Programming at the IT organization level). Our framework allows for the integration of these CBDMs in one EA project. It can be used as a basis that proposes a uniform way for representing concerns at all organization levels: it proposes a unique terminology that allows a modeler to align different CBDMs used at different organization levels.

Future work

The main problem related to using our framework in "industry scale" projects is related to the amount of manual ("paper") work that that a modeler has to deal with when he defines and composes roles in his models. Therefore, the main need is to provide a tool that can support the modeling method proposed in this thesis. The diagrams that represent the result of the composition should be automatically generated based on the base roles and composition constraints. However, without the tool this work should be done manually. This takes time, introduces "copy/paste" mistakes and inconsistencies. As a result, our framework can not be fully used to build large, "industrial scale", models. For example, in the third application of our framework (Chapter 8) we did not aim to use our modeling language to represent all roles in a manufacturing process. Instead we provided an approach that allows a modeler to understand the rationale behind each action of a manufacturing process by means of thinking in terms of roles.

The availability of a tool will help a modeler to use our framework for the "industry scale" projects. The future of this thesis is mainly related to the implementation of the case tool. The most plausible way to build this tool is to extend the existing SeamCAD tool[26]. This tool implements the Systemic Enterprise Architecture Methodology (SEAM) that was developed in our laboratory. To extent the SeamCAD tool, our situation-based modeling framework should be integrated with SEAM. We see the following issues related to the development of a tool and to the integration of our framework with SEAM:

- Extend the existing SEAM meta-model for the situation-based modeling: the hierarchy of situations and corresponding roles should be supported. Note that in our framework multiple views can be defined for the same system. This means that the system can have multiple role/situation hierarchies;
- Extend the existing visual language of the SeamCAD tool with the elements that support multi-situational modeling;

[26] See the description of this tool on the following web-page: http://lamspeople.epfl.ch/le/

- Define a library of composition constraints as the part of the meta-model mentioned above;
- Define a mechanism that allows a modeler to compose roles/situations automatically based on composition constraints;
- Consider possible transformations (export/import) of situation-based models into the existing CBDMs tools, such as RADRunner (see http://www.rolemodellers.com). This will allow SeamCAD to become a tool for the integration of different CBDMs;
- Consider to extend the SeamCAD tool to document the design rationale history. In Chapter 8 we have shown how our method can be used for capturing the design rationale in modeling manufacturing processes.

The availability of a tool can certainly make the Situation-Based Modeling Framework more interesting for large projects. However, the tool is only one side of the problem. The other side of the problem is to find directions where our framework can have the most positive results. Currently the most prominent direction is to use our Situation-Based Modeling Framework for the creation of the modeling templates. Below we explain this direction in more details.

Often a modeling process starts from the scratch (from a white piece of paper). This can be quite frustrating for junior modelers (business annalists or software architects) who do not know where to start. Our framework can help to solve this problem by suggesting initial modeling steps:

- Identify situations and objects that participate in these situations. Base situations (at the lowest level of the situation hierarchy) should be related to the management of information objects that a system of interest has to support. In the example with a simple banking system (see Chapter 4), such information object is an account that the banking system has to keep.
- Specify roles that correspond to basic situations and make explicit the life cycle of the corresponding information objects (such as making explicit the following actions in the Control Account role: Create Account, Make Transaction and Delete Account).
- Compose the base roles/situation into the hierarchy of roles/situations.

Note that this process can be automated. For example, when a modeler specifies the information objects that the system of interest should support, the case tool can create all actions (CreateX, MakeX and DeleteX) that are related to the life cycles of these objects. The process of composition of situations/roles can also be automated, as we explained earlier in this thesis.

Modeling templates can be defined for different organization levels in EA. For example, at the IT organization level it can be interesting to define base roles for J2EE-based projects: the role a programmer, the role of J2EE architect, the role of a deployer, etc. At the business related organization levels, our framework can be used to define role templates for business process models. The definition of these role templates would be the most notable in business domains that include many business stakeholders, multiple business goals and activities (for example, the domain of manufacturing). In this thesis we have considered some base roles (such raw material provider, quality control and product manufacturing roles) for modeling manufacturing processes. In the future work the model of these roles can be further refined and other roles can be specified.

An interesting field where our framework can be useful is Service Oriented Architecture (SOA). SOA considers systems as collections of services. A service is a well-defined function provided by a service provider for a service consumer. This is quite similar to our approach where we consider systems as compositions of roles. Roles define functionality for other roles in collaborations. This similarity allows us to use our framework for the high-level specification of services in SOA. Currently there are many approaches that pay attention to the formal side of the SOA. Such approaches study the formal definition and the composition of services. However, such formal approaches for SOA cannot be used for the specification of services at the business level: these approaches are usually too complex. Our framework can be used for this purpose. In particular our framework allows a modeler:

- To specify services graphically in the form of role hierarchy;
- To make explicit how services are composed (or orchestrated);
- To make explicit composition constrains between services;
- To make explicit the management of enterprise entities (how enterprise entities are created, persisted and deleted).

Diagrams such as in Figures 19, 20 (Section 4.2) are particularly interesting for SOA: they can be used to specify how multiple base services are composed into one bigger service. Such diagrams explicitly show the management of enterprise information objects (or entities) and orchestration of roles.

REFERENCES

[AIAA98] AIAA – American Institute of Aeronautics and Astronautics, "Glossary of Verification and Validation Terms", 1998, retrieved from http://www.grc.nasa.gov/WWW/wind/valid/tutorial/glossary.html on April 2004

[Achermann01] Achermann, F., et al., Piccola - a Small Composition Language, in Formal Methods for Distributed Processing - A Survey of Object-Oriented Approaches, H. Bowman and J. Derrick, Editors. 2001, Cambridge University Press. p. 403--426.

[Aksit88] Aksit M., Tripathi, A., Data Abstraction Mechanisms in Sina/st, Proceedings of the conference Object-Oriented Systems, Languages and Applications (OOPSLA'88), ACM Sigplan Notices, Vol. 23, No. 11, pp. 267-275, San Diego, September 1988.

[Aksit94] Aksit, M., K. Wakita, J. Bosch, L. Bergmans and A. Yonezawa, "Abstracting Object Interactions Using Composition Filters In object-based distributed processing, R. Guerraoui, O. Nierstrasz and M. Riveill (Eds.), LNCS, Springer-Verlag, pp. 152-184, 1993.

[Aldawud01] O. Aldawud, T. Elrad, and A. Bader, "A UML Profile for Aspect Oriented Modeling," presented at Workshop on AOP at OOPSLA 2001, Tampa Bay, FL, USA, 2001

[Alexander77] Alexander, C., S. Ishikawa, and M. Silverstein, "A Pattern Language: Towns, Buildings, Construction", 1977: Oxford University Press, ISBN 0195019199

[Andrade99] Andrade, L. and J. Fiadeiro, "Interconnecting Objects via Contracts", in Proc. UML'99, Springer Verlag, 1999

[Andrade01] Andrade, L. and J. Fiadeiro. "Coordination Technologies for Managing Information System Evolution", in Proceedings of CAiSE'01, Interlaken, Switzerland: Springer-Verlag, 2001

[Appleton98] Appleton, B., "Patterns and Software: Essential Concepts and Terminology", 1998, accessed from "http://www.cmcrossroads.com/bradapp/docs/patterns-intro.pdf" on October 3, 2003

[Atkinson99] C. Atkinson, T. Kühne, and C. Bunse, "Dimensions of Component-based Development," presented at Fourth International Workshop on Component-Oriented Programming (WCOP'99; in conjunction with ECOOP'99), 1999.

[Audi99] Audi R., "The Cambridge Dictionary of Philosophy", 2nd edition ed., Cambridge University Press, 1999, ISBN 0521637228

[Balabko03] Balabko, P., Wegmann, A., "A Composition of Business Role Models", Proc. of ICEIS conference, Angers, France, 2003

[Balabko03a] P. Balabko and A. Wegmann, "Context Based Reasoning in Business Process Models," Proc. of IEEE Information Reuse and Integration (IRI 2003), Las Vegas, USA, 2003

[Balabko03b] Balabko P., A. Wegmann "From RM-ODP to the Formal Behavior Representation", book chapter in "Practical Foundations of Business and System Specifications", Kluwer Academic Publishers, 2003

[Barbier03] Barbier, F., et al., "Formalization of the Whole-Part Relationship in the Unified Modeling Language", IEEE Transactions on Software Engineering, 2003, 29(5): p. 459-470.

[Barwise83] Barwise J., J. Perry, "Situations and Attitudes", Cambridge, MA: MIT Press, 1983.

[Banathy96] B. Banathy, "A TASTE OF SYSTEMICS", 1996, retrieved from http://www.isss.org/taste.html on November 10, 2003

[Banathy] B. Banathy, "THE EVOLUTION OF SYSTEMS INQUIRY", retrieved from http://www.isss.org/primer/003evsys.htm on November 13, 2003

[Berger02] Berger, J., "The value of outsourcing", whitepaper, DecisionOne Corporation, retrieved from: http://www.decisionone.com/d1m/news/white_papers, 2000

[Bernstein99] Bernstein, A., Klein, M., & Malone, T.W., "The Process Recombinator: a Tool for Generating New Business Process Ideas", International Conference on Information Systems, 1999

[Bækdal99] Bækdal, L., K., and B. B. Kristensen, "Aggregation from Multiple Perspectives by Roles," presented at IEEE International Conference on Technology of Object-Oriented Languages and Systems (TOOLS PACIFIC 99), Melbourne, Australia, 1999.

[Bouquet01] Bouquet, P., et al., “Theories and Uses of Context”, Knowledge Representation And Reasoning, 2001, technical report, Centro Per La Ricerca Scientifica e Tecnologica: Povo (Trento), retrieved from http://www.itc.it

[Broy91] Broy, M., Formal treatment of concurrency and time, in Software Engineer's Reference Book, J. McDermid, Editor. 1991, Oxford: Butterworth-Heinemann,. p. 23/1-23/19.

[Buckingham96] Buckingham Shum, S. (1996). The Encyclopedia of Computer Science and Technology (Marcel Dekker Inc: NY), Vol. 35, Supp. 20, 95-128.

[Bunge79-4] Bunge, M., “Ontology II: A World of Systems”, “Treatise on Basic Philosophy” series, Vol. 4. 1979: D. Reidel Publishing.

[Burnett01] Burnett M., “Software Engineering for Visual Programming Languages”, Handbook of Software Engineering and Knowledge Engineering, 2001, World Scientific Publishing Company.

[Carlsen97] Carlsen, S., “Conceptual Modeling and Composition of Flexible Workflow Models”, Norwegian University of Science and Technology, PhD Thesis, 1997.

[Champeau03] Champeau, J., F. Mekerke, and E. Rochefort, “Towards a Clear Definition of Patterns, Aspects and Views in MDA”, in proc. of Engineering Methods to Support Information System Evolution Methods workshop, 2003, Geneva, Switzerland

[Chang99] Chang, S.K., et al. “The Future of Visual Languages”, Proc. IEEE Symposium on Visual Languages, 1999, Tokyo, Japan.

[Checkland99] Checkland, P., “Systems Thinking, Systems Practice”, Chichester, UK: Wiley, 1999.

[Collins-Cope98] Collins-Cope, M., “Object Oriented Analysis and Design using UML”, white paper, 1998, Ratio Group Ltd.: London, retrieved from http://www.ratio.co.uk/white.html

[Conklin89] Conklin, J. and Begeman, M.L., “gIBIS: A Tool for All Reasons”, Journal of the American Society for Information Science, May: 200-213, 1989

[Conklin91] Conklin, J. and Burgess Yakemovic, K.C., “A Process-Oriented Approach to Design Rationale”, Human-Computer Interaction, 6, 3&4: 357-391, 1991.

[Cyril97] L. Cyril and S. Catherine, Some Remarks on Wholes, Parts and Their Perception, Psycoloquy electronic journal (ISSN 1055-0143), 1997, accessed from "http://psycprints.ecs.soton.ac.uk/archive/00000549/" on August 20, 2003

[Czarnecki00] Czarnecki K., Eisenecker U. “Generative Programming Methods, Tools, and Applications”, Addison-Wesley Pub Co; 1st edition, June, 2000

[Davenport90] Davenport, T.H., Short, J.E., “The New Industrial Engineering: Information Technology and Business Process Redesign”, Sloan Management Review, 1990

[Dembski94] Dembski, W.A., “The Fallacy of Contextualism”, The Princeton Theological Review, 1994, retrieved from http://www.arn.org/docs/dembski/

[Dori00] Dori, D., “Object-Process Methodology. A Holistic Systems Paradigm”., Heidelberg, New York: Springer Verlag, 2000

[D'Souza98] D'Souza, D.F. and A.C. Wills, Objects, Components, and Frameworks With Uml: The Catalysis Approach. Addison-Wesley Object Technology Series. 1998: Addison-Wesley Pub Co.

[Deutsch02] H. Deutsch, "Relative Identity", Stanford Encyclopedia of Philosophy, 2002, retrieved from http://plato.stanford.edu/entries/identity-relative/ on January 20, 2004

[Fiadeiro97] Fiadeiro, J. L. and A. Lopes, “Semantics of Architectural Connectors”, in Proc. of TAPSOFT'97, Springer-Verlag, 1997

[Fiadeiro01] Fiadeiro, J., Lopes, A., & Wermelinger, M., “A Mathematical Semantics of Architectural Connectors”, ATXSOFTWARE, retrieved from: http://www.atxsoftware.com/publications.html, 2001

[Gamma94] Gamma, E., et al., “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley Professional Computing Series, 1994: Addison Wesley Publishing Company, ISBN 0201633612

[Gauze02] Gauze, C. F., “Fanning unveils failed Napster business model!”, Ink 19, Retrieved from http://www.ink19.com/issues/april2002/features/, April, 2002

[Gerstl96] Gerstla P. and S. Pribbenow, "A conceptual theory of part-whole relations and its applications," Data & Knowledge Engineering, vol. 20, pp. 305-322, 1996.

[Genilloud00] Genilloud, G., & Wegmann. A., “A Foundation for the Concept of Role in Object Modeling”, EDOC, 2000.

[Giunchiglia93] Giunchiglia, F., “Contextual Reasoning”, Technical Report #9211-20, University of Trento, 1993.

[Gubina96] V.D.Gubina, T.J.Sidorina, V.P. Philatova, “Philosophy: Textbook” in Russian, Russian Word, 1996. p 432

[Halpin01] Halpin, T., "Object Role Modeling: An Overview", Microsoft Corporation, 2001, retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vstchvsea_ormoverview.asp on February 16, 2004

[Harrison93] Harrison, W. and H. Ossher, “Subject-oriented programming (a critique of pure objects)”, in Proc. of OOPSLA'93, 1993

[Holt83] Holt A. W., H. R. Ramsey, and J. D. Grimes, "Coordination System Technology as the basis for a programming environment," Electrical Communication, vol. 57(4), 1983

[Huibers96] Huibers T. W. C., M. Lalmas, and C. J. v. Rijsbergen, "Information retrieval and situation theory," in ACM SIGIR Forum, vol. 30: ACM Press, 1996, pp. 11 - 25.

[IBM03] http://www.alphaworks.ibm.com/tech/hyperj, accessed on November 24, (2003)

[ISO96] ISO/IEC, 10746-1, 3,4 | ITU-T Recommendation X.902, “Open Distributed Processing - Basic Reference Model - Part 2: Foundations”, 1996

[Jackson93] Jackson M., P. Zave, “Domain Descriptions”, Proc. RE’93, 1st Intl. IEEE Symposium on Requirements Engineering, Jan. 1993, 56-64

[Jackson00] Jackson, D., Alloy: A Lightweight Object Modeling Notation, Technical Report 797, 2000, MIT Laboratory for Computer Science: Cambridge, MA.

[Jackson02] Jackson, D., “Micromodels of Software: Lightweight Modelling and Analysis with Alloy”, Software Design Group, MIT Lab for Computer Science, 2002, downloaded from http://sdg.lcs.mit.edu/alloy/reference-manual.pdf

[Jensen95] Jensen, M. and M. D. Jørgensen, "Roles -- A Dynamic Alternative (in Danish)" Aalborg University, 1995.

[Kandé00] Kandé M. M. and A. Strohmeier, "Towards a UML Profile for Software Architecture," in Proc. of UML'2000 - The Unified Modeling Language: Advancing the Standard, York, UK, 2000.

[Kandé01] M. M. Kandé, "ConcernBASE: Architecture-Centered Software Engineering", retrieved from http://lgl.epfl.ch/research/concernbase/index.html on 20.02.2004, 2001

[Kande02] M. M. Kande, et al., "From AOP to UML - A Bottom-Up Approach", in proc. of the Aspect-Oriented Modeling with UML workshop, Enschede, The Netherlands, 2002

[Kendall98] E. A. Kendall, "Goals and Roles: The Essentials of Object Oriented Business Process Modeling," presented at European Conference on Object Oriented Programming (ECOOP) Workshop on Object Oriented Business Process Modeling, 1998.

[Kendall99] Kendall, E.A. “Role Model Designs and Implementations with Aspect Oriented Programming”, in Proc. of the 1999 Conference on Object- Oriented Programming Systems, Languages, and Applications (OOPSLA'99). 1999: ACM Press.

[Kiczales97] Kiczales, G., et al., “Aspect-Oriented Programming”, in Proc. of European Conference on Object-Oriented Programming, Berlin, Heidelberg, and New York 1997

[Kilov99] Kilov, H., Business Specifications: The key to successful software engineering. 1999, Upper Saddle River, NJ: Prentice-Hall.

[Kristensen95] Kristensen, B. B., "Object-Oriented Modeling with Roles," presented at 2nd International Conference on Object-Oriented Information Systems, 1995.

[Kueng96] Kueng, P., et al., “How to compose an Object-Oriented Business Process Model?”, in Proc. of IFIP Conference on Method Engineering, 1996

[Lakoff87] Lakoff, G., “Women, Fire and Dangerous Things – What Categories Reveal about the Mind”, Chicago Press, 1987, ISBN 0-226-46804-6

[Lamport90] Lamport, L. and N.A. Lynch, Distributed Computing: Models and Methods, in Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. 1990, Elsevier and MIT Press.

[Lee90] Lee, J., “SIBYL: A Tool for Managing Group Design Rationale”, in Computer Supported Cooperative Work, Los Angeles, CA, ACM Press: New York, pp. 79-92, 1990.

[Lee91] Lee, J. and Lai, K., “What’s in Design Rationale?”, Human-Computer Interaction, 6, 3&4: 251-280, 1991.

[Leite91] Leite, J.C.S.d.P. and P.A. Freeman, Requirements Validation through Viewpoint Resolution. IEEE Transactions of Software Engineering, 1991. 17(12).

[Lieberherr92] Lieberherr K. J., "Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes," presented at Information Processing '92, 12th World Computer Congress, 1992., pp. 179-185.

[LinguaLink03] LinguaLink Library, Version 5.0 published on CD-ROM by SIL International, 2003, accessed from http://www.ethnologue.com/ll_docs/contents.asp on April 19, 2004.

[Lupu95] E. C. Lupu and M. S. Sloman, "An Approach to Role Based Management for Distributed Systems," Imperial College, London, Research Report DOC 95/9, 1995.

[MacLean91] MacLean, A., Young, R.M., Bellotti, V. and Moran, T., "Questions, Options, and Criteria: Elements of Design Space Analysis", Human-Computer Interaction, 6, 3 & 4: 201-250, 1991

[Malhotra98] Malhotra, Y., "Business Process Redesign: An Overview", IEEE Engineering Management Review, vol. 26, no. 3, 1998

[Marcaillou94] Sophie Marcaillou, Abdelaziz Kriouile, Bernard Coulette, "VBOOL, une extension d'Eiffel intégrant le concept de point de vue", in proceedings of Magrebian Conference on Software Engineering and Artificial Intelligence MCSEAI'94, pp 115-125, Rabat, 11-14 avril 1994

[Mark87] Mark L., "Defining views in the binary relationship model," Information Systems, vol. 12, pp. 281 - 294, 1987

[Mayer92] Mayer R. J., M. Painter and P. deWitte, "IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications", Knowledge Based Systems, Inc. 1992

[Mayer95] Mayer R. J., John W. Crump, IV, Ronald Fernandes, Arthur Keen, Michael K., "Painter Information Integration for Concurrent Engineering (IICE): Compendium of Methods Report", report, COMMAND WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7604, 1995

[Motschnig-Pitrik99] Motschnig-Pitrik, R., "Contexts and Views in Object-Oriented Languages.", in proc. of IEEE conference CONTEXT 1999, Lecture Notes in Computer Science, vol 1688, Trento, Italy

[Motschnig-Pitrik99a] Motschnig-Pitrik R. and J. Kaasbøll, "Part-Whole Relationship Categories and Their Application in Object-Oriented Analysis," IEEE Transactions on Knowledge and Data Engineering, vol. 11, 1999.

[Motschnig-Pitrik00] Motschnig-Pitrik, R., "Contexts as means to decompose Information Bases and represent relativized Information", Proc. CHI Workshop #11: The Who, What, Where, When, Why and How of Context-Awareness, 2000, Hague, Netherlands.

[Motschnig-Pitrik00a] Motschnig-Pitrik R., "A Generic Framework for the Modeling of Contexts and its Applications", Data & Knowledge Engineering, vol. 32, pp. 145-180, 2000.

[Motschnig-Pitrik00b] Motschnig-Pitrik R., "The Viewpoint Abstraction in Object-Oriented Modeling and the UML", Proc of ER 2000, Salt Lake City, Utah, 2000.

[Miers01] Miers, D., "Modern Business Strategies and Process Support", white paper, Enix Consulting Ltd, England, 2001

[Miller95] Miller J.G., "Living Systems", University of Colorado Press, 1995

[Milner99] Milner, R., "Communicating and Mobile Systems: the pi-Calculus", Cambridge University Press, 1999

[Mylopoulos95] Mylopoulos J. and R. Motschnig-Pitrik, "Partitioning Information Bases with Contexts," presented at International Conference on Cooperative Information Systems, Vienna, 1995.

[Nassar03] Nassar, M., et al., "Towards a View Based Unified Modeling Language", in proc. of 5th International Conference on Enterprise Information Systems (ICEIS), 2003, Angers, France.

[Naumenko01] Naumenko, A., et al. A Viewpoint on Formal Foundation of RM-ODP Conceptual Framework, Technical report No. DSC/2001/040, July 2001, EPFL-DSC ICA

[Naumenko02] Triune Continuum Paradigm: a paradigm for General System Modeling and its applications for UML and RM-ODP, Ph.D thesis number 2581, EPFL June 2002.

[Nentwich03] Nentwich C., W. Emmerich, and A. Finkelstein, "Consistency Management with Repair Actions," presented at IEEE International Conference on Software Engineering, Portland, Oregon, 2003.

[NIA2002] Northern Illinois University, Victoria TelecommunityNet, "Glossary of Communications, Computer, Data, and Information Security Terms", accessed on April 27, 2004 from http://sun.soci.niu.edu/~rslade/secgloss.htm

[NIST93] NIST, "IDEFØ - standard for Function Modeling in FIPS Publication 183", NIST, (1993), retrieved from http://www.idef.com/idef0.html on March 20, 2004

[Nuseibeh93] B. Nuseibeh, J. Kramer, and A. Finkelstein, "Expressing the Relationships between Multiple Views in Requirements Spec," in Proc. of ICSE, 1993.

[Odeh02] M. Odeh, I. Beeson, S. Green, and J. Sa, "Modelling Processes using RAD and UML Activity Diagrams: an Exploratory Study," in proc. of International Arab Conference on Information Technology (ACIT'2002), Doha Qatar, 2002.

[OMG01] OMG, Unified Modeling Language Specification, v 1.4, 2001, Version 1.4

[OpenGroup03] The Open Group, "The Open Group Architecture Framework version 8 (TOGAF)," 2003.

[Opie99] J. Opie, "Gestalt Theories of Cognitive Representation & Processing," Psycoloquy, electronic journal, vol. 10, 1999, downloaded from http://psycprints.ecs.soton.ac.uk/archive/00000656/

[Ossher00] H. Ossher and P. Tarr., Multi-Dimensional Separation of Concerns and The Hyperspace Approach, Kluwer, 2000, accessed from "http://citeseer.nj.nec.com/ossher00multidimensional.html" on 29.09.2003

[Ould95] Ould M. A., “Business Processes: Modeling and analysis for re-engineering and improvement”, John Wiley & Sons, Chichester, 1995

[Pinheiro02] Pinheiro, F. A. C. and I. D. López, "Writing Use Cases Modelled with Situation Theory," presented at Workshop on Requirements Engineering (WER 2002), Valencia, Spain, 2002

[Phalp97] Phalp K.T. (1997). “Using Counts as Heuristics for the Analysis of Static Models”, Workshop on Process Modeling and Empirical Studies of Software Engineering, 1997

[Pnambic99] "Compositional Program Composition - Related Work", Pnambic Computing, 1999, retrieved from http://www.pnambic.com/CPS/Related.html on April 20, 2004

[Poincaré83] Poincaré H, The value of science, Moscow «Science», 1983

[Rea04] Rea, M. C., "Relative Identity and the Doctrine of the Trinity.," Philosophia Christi, forthcoming, retrieved from http://www.nd.edu/~mrea/papers.html on 21.01.2004

[Reenskaug96] Reenskaug, T., et al., “Working With Objects: The OOram Software Engineering Method”, ed: Manning Publication Co, 1996

[Regev01] Regev, G. & Wegmann, A., “Goals, Interpretations, and Policies in Information Systems Design”, EPFL-IC-LAMS, Technical report No. DSC/2001/043, retrieved from http://icawww.epfl.ch/, 2001

[Rescher55] Rescher, N. and P. Oppenheim, Logical analysis of Gestalt concepts. Brithish Journal for the Philosophy of Science, 1955(VI (22)): p. 89—106

[Riehle96] Riehle. D., "Describing and Composing Patterns Using Role Diagrams," presented at WOON’96, Russia, St. Petersburg, 1996.

[Riehle97] Riehle, D., "Composite Design Pattern" presented at Conference of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'97), 1997.

[Riehle98] Riehle, D. and T. Gross. “Role Model Based Framework Design and Integration”, in Proc. of Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'98): ACM Press

[Rychkova03] Rychkova I., A. Wegmann, P. Balabko, "Operational ASM Semantics behind Graphical SEAM Notation", Proceedings of DAIS/FMOODS Ph.D. workshop, Paris, November 2003, pp. 10

[Schünemann01] U. Schünemann, Truth and Reality, Department of Computer Science, Memorial University of Newfoundland, 2001, accessed from "http://web.cs.mun.ca/~ulf/gloss/real.html" on 12.05.03

[Schünemann01a] U. Schünemann, Wholes & System Composition Levels, Department of Computer Science, Memorial University of Newfoundland, 2001, accessed from "http://web.cs.mun.ca/~ulf/gloss/wholes.html" on 11.12.03

[Singh92] Singh, B., Rein, G.L., “Role Interaction Nets (RINs): A Process Description Formalism”, Technical Report CT-083-92, Microelectronics and Computer Technology Corp., July 1992

[Skobeltsyn03] Skobeltsyn G., Balabko P., “RaP tool: one possible scenario” accessed from http://lamspeople.epfl.ch/balabko/RaPTool/ on November 24, (2003)

[Sowa84] Sowa J.F., “Conceptual Structures: Information Processing in Mind and Machine”, Addison-Wesley, New York; 1984

[Sowa99] Sowa J. F., Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove, CA: Brooks Cole Publishing Co., 1999.

[Stähler02] Stähler, P., “Business Models as a Unit of Analysis for Strategizing”, International workshop on business models, Lausanne, UNIL, 2002

[Steimann00] Steimann, F., "On the Representation of Roles in Object-Oriented and Conceptual Modeling," Data and Knowledge Engineering 35, pp. 83–106, 2000

[Steimann03] Steimann F., J. Gößner, and T. Mück, "On the Key Role of Composition in Object-Oriented Modelling" presented at UML 2003: Modeling Languages and Applications, San Francisco, California, USA, 2003

[Stein02] D. Stein, S. Hanenberg, and R. Unland, "A UML-based aspect-oriented design notation for AspectJ," presented at First Conference on AOSD, Enschede, The Netherlands, 2002

[Stevens00] Stevens, P. and R. Pooley, Using UML Software Engineering with Objects and Components (Updated Edition). Object Technology Series. 2000

[Suzuki99] J. Suzuki and Y. Yamamotto, "Extending UML with Aspects:Aspect Support in the design phase", presented at 3rd Aspect-Oriented Programming Workshop at ECOOP, 1999

[Tarr00] Tarr P., Ossher H., Hyper/J™ User and Installation Manual, included in the hyperj.zip file downloaded from http://www.alphaworks.ibm.com/tech/hyperj, 2000.

[Tinggard95] Tinggard, C., and T. Worm, "Modeling and Programming Languages (in Danish)," Aalborg University, 1995

[Truyen00] Truyen, E., B.N. Jørgensen, and W. Joosen, “Customization of Component-based Object Request Brokers through Dynamic Reconfiguration”, in Proc. of IEEE TOOLS EUROPE conference, 2000, Mont Saint-Michel, France

[Turner98] Turner, C.R., et al, “Feature Engineering”, in Proc. of IEEE 9th International Workshop on Software Specification and Design, 1998, Ise-Shima (Isobe), Japan

[VanHilst96] VanHilst, M. and D. Notkin. “Using Role Components to Implement Collaboration-Based Designs”, in proc. OOPSLA'96, 1996: ACM Press

[Varzi03] Varzi, A., "Mereology", Stanford Encyclopedia of Philosophy, 2003, retrieved from http://plato.stanford.edu/entries/mereology/ on April 7, 2004

[VanHilst96] M. VanHilst and D. Notkin, "Using Role Components to Implement Collaboration-Based Designs", presented at OOPSLA'96, 1996

[Vassallo01] Vassallo, N., “Contexts and Philosophical Problems of Knowledge”, Proc. CONTEXT conference, 2001, Springer-Verlag.

[Wegmann01] Wegmann, A. and A. Naumenko. Conceptual Modeling of Complex Systems Using an RM-ODP Based Ontology. in 5th IEEE International Enterprise Distributed Object Computing Conference - EDOC 2001. 2001. Seattle, ACTION.

[Wegmann03] Wegmann, A. “On the systemic enterprise architecture methodology (SEAM)”, Proc. of International Conference on Enterprise Information Systems (ICEIS 2003), Angers, France, 2003

[Wegmann03a] Wegmann, A., O. Preiss, "MDA in Enterprise Architecture? The Living System Theory to the Rescue..." Proc. of Enterprise Distributed Object Computing Conference (EDOC'03), Brisbane, Queensland, Australia, 2003

[Wegmann04] Wegmann Alain, Balabko Pavel, Le Lam-Son, Regev Gil, Rychkova Irina, “A Method and Tool for Business-IT Alignment in Enterprise Architecture”, will be published in proceedings of the CAiSE forum 2005, Porto, Portugal.

[Wilson93] Wilson, B. G., Jonassen, D. H., Cole, P., “Cognitive approaches to instructional design”, in The ASTD handbook of instructional technology, New York: McGraw-Hill, 1993

[Wing90] Wing, J.M., “A Specifier's Introduction to Formal Methods”, IEEE Computer, 23(9): p. 8-24, 1990

[Wisse01] P. Wisse, "METAPATTERN: information modeling as enneadic dynamics", University of Amsterdam, 2001, retrieved from http://primavera.fee.uva.nl/html/working_paper_details.cfm?Id=118 on January 27, 2004

[Wisse00] P. Wisse, "Metapattern: Context and Time in Information Models”, Addison-Wesley Longman Publishing Co, 1st edition, 2000.

[WMC95] Workflow Management Coalition, “The Workflow Reference Model”, retrieved from: http://www.wfmc.org/standards/standards.htm, January 1995

[Yu94] Yu, E., and Mylopoulos, J. "Using Goals, Rules and Methods to Support Reasoning in Business Process Reengineering," Proc. Int. Conf. System Sciences, Hawaii, pp. 234-243, 1994

[Zachman87] Zachman J.A., “A Framework for Information Systems Architecture”, IBM System Journal, vol. 26, issue 3, pp. 276 – 292, 1987

APPENDIX 1
LINK TO THE IMPLEMENTATION

Our framework is a conceptual framework that aims to improve conceptual modeling in the context of EA. As we explained earlier in Chapter …, in this thesis we are interested in modeling the information viewpoint of computational objects, i.e. we are interested in modeling computational objects in terms of roles that these objects play in different situations. Our work is limited to modeling at one given level of the hierarchy of organizational levels. We do not aim to define how different organization levels are related to each other, i.e. how objects at one organization level are refined (or "implemented") at the lower organization levels. This topic is a subject of a separate research work.

However, in order to improve the overall understanding of our framework, we would like to give some basic ideas about the refinement of the organizational levels. When an object is refined into the set of collaborating objects at the lower-level of the organizational hierarchy the following transformations can be applied to the objects' roles:

- A single role (at the level *n)* is split into several roles (at the level n+1) to be played by different computational objects. For example, in Figure 54.a the A role at the Company organization level is split between three objects: the attributes of the A role would typically go into a Data Base Server, the behavior of the A role would be distributed between the Business Logic and Presentation Logic servers. The interfaces between these three objects should be defined in order to provide interactions between the corresponding sub-roles. The design decisions about splitting the A role depend very much on the used technology and the experience of the system architect that have to do this work. The choice of the technology and the design decisions should take into consideration the Quality of Service (QoS) requirements that can be specified additionally to the pure functional specification in forms of roles. The QoS is not included in our framework[27] (that is why we show the QoS in Figure 54 in a separate box). QoS can be specified simultaneously with the pure functional specifications that we propose in our framework. The refinement decisions (such as in Figure 54.a) about splitting roles between several computational objects should guarantee that the QoS requirements are satisfied.
- Several roles have to be integrated (or composed into a single role) in order to be played by a single computational object (Figure 54.b). For example, in Figure 54.b the A.1 and A.2 have to be integrated for putting them in the common execution context of the Business Logic Server.

[27] The question about QoS is also related to the question about the scalability. We did not address the scalability problem in our framework. This is a separate problem. The question of scalability in the same way as QoS is related to the problem of refinement of organization levels, i.e. how a functionality of system, specified using our framework, is realized with a given technology (such as three-tiered architecture).

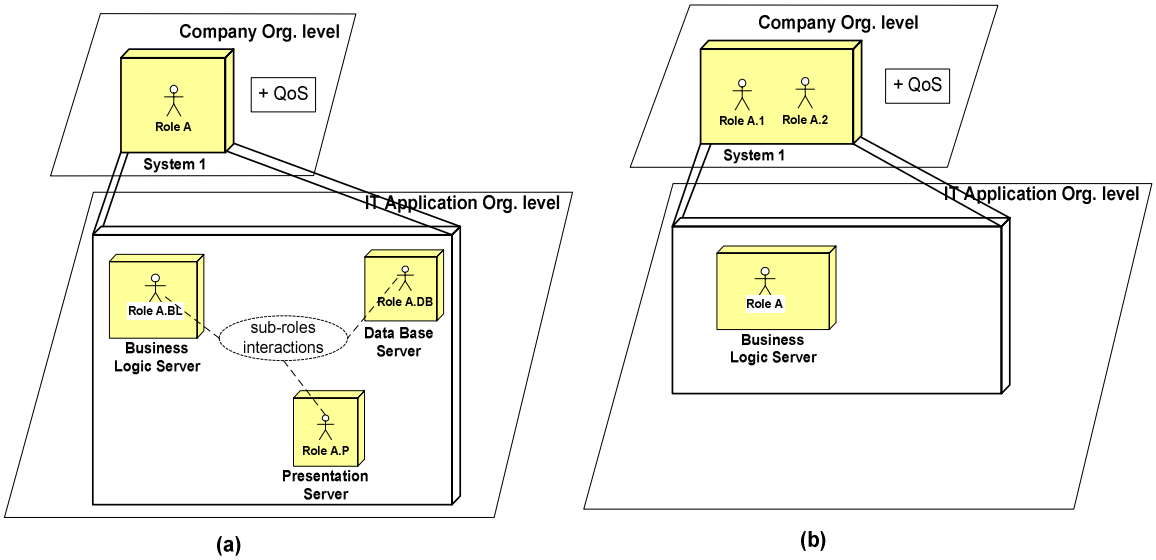


Figure 54. Relation between organization levels; a: Decomposition of the Role A; b. Composition of roles A.1 and A.2

Almost any project has to deal with both aforementioned role transformations (composition and decomposition). For example, in the example with the Simple Banking System, each base role (such as "Create Account" and "Make Transaction" roles in Figure 55) has to be decomposed between available computational objects. As we have already explained above this decomposition is based on the technological choice and the experience of the system architect.

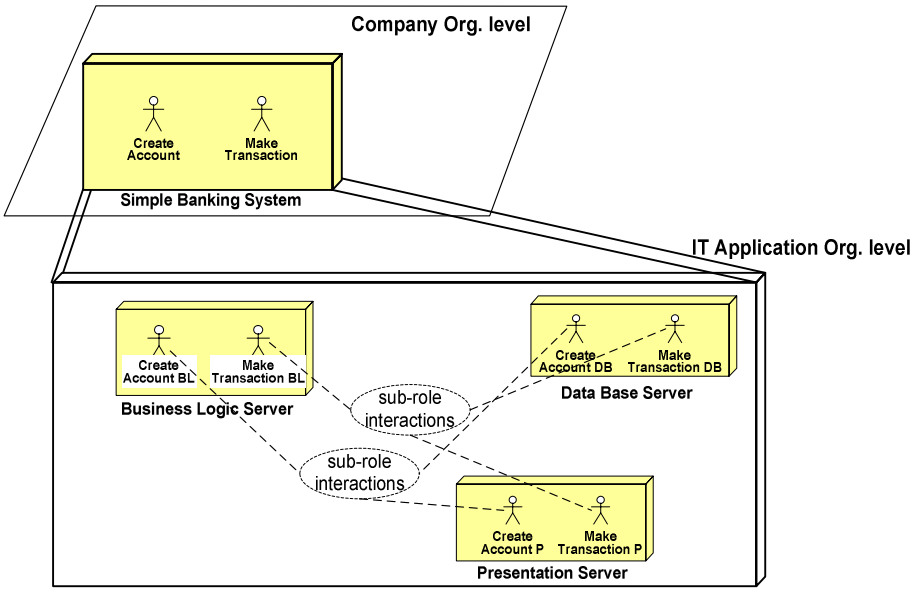


Figure 55. Role transformations: roles are decomposed into sub-roles; these sub-roles have to be integrated in the context of each computational object

The described decomposition of roles can be continued to the lower-levels (such as component level) of the organizational hierarchy. However at the the lowest, « implementation », level the base roles have to be integrated (or composed) in the context of each computational object. For example, the "Create Account DB" and "Make Transaction DB" roles have to be integrated in the execution context of a Data Base Server.

Below we consider how the aforementioned integration (or composition) of base roles can be achieved at the implementation level. Many different methods can be adapted for the composition of base roles that have to be put in a common execution context (see the list of methods in the "Programming and Modeling for Programming" sub-section of Section 2.1). In our work we give some general suggestions that can help designers to choose the appropriate methods for the composition of base roles. Between all methods that can be

used for the implementation of role compositions we can distinguish two implementation patterns, i.e. these methods can be divided into two groups:

- Methods that compose roles by merging the specifications of base roles into a specification of a new role. We call this "Merging Base Roles" implementation pattern.
- Methods that compose base roles by means of coordinating their behavior. We call this "Coordinating Base Roles" implementation pattern.

Further, we use the example of the aforementioned Simple Bank to show more details about each implementation pattern.

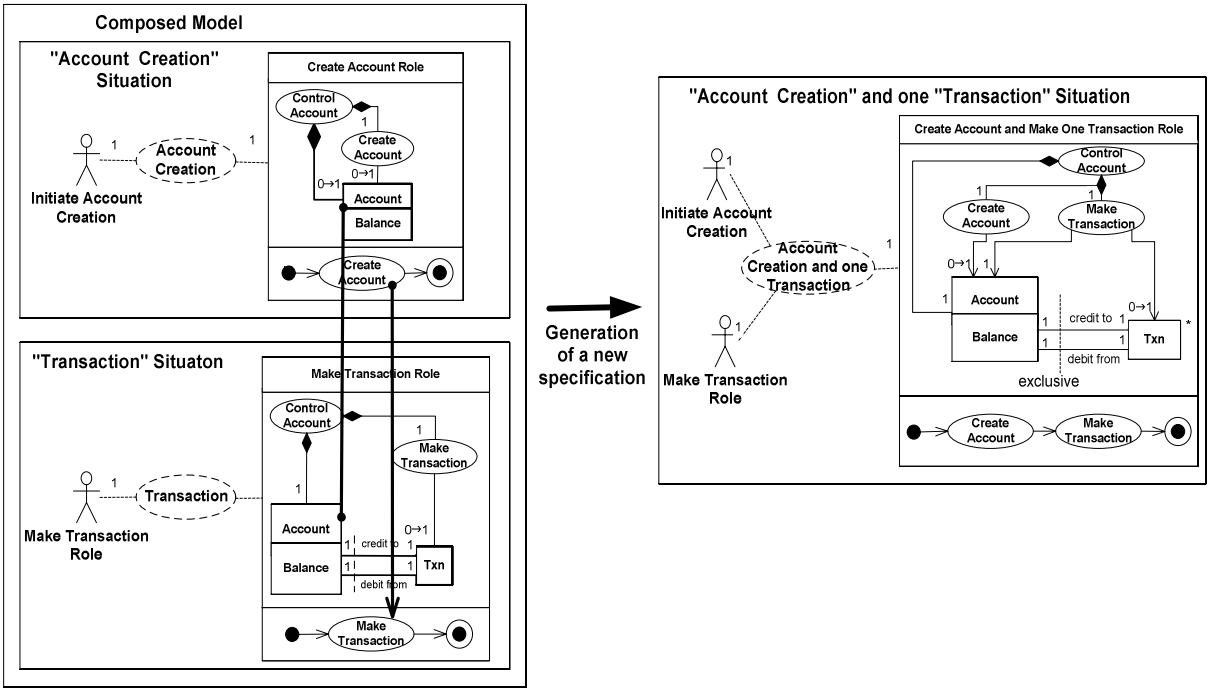


Figure 56. Implementation patterns:
Role Model Composition Specification and its implementations: Merging Base Roles

The first implementation pattern, *"Merging Base Roles*", merges the specification of base roles into the specification of a new role ("Create Account and Make One Transaction" role in Figure 56). This new role is entirely responsible for carrying out the composition constraints. In this implementation pattern, the concept of a role is used as a design tool: roles are intermediate specifications that are composed together to form a final specification for each computation object. When all roles have been composed, the final specification can be implemented.

Note that in the "Merging Base Roles" implementation pattern, once roles are composed into a new role, this new role is assigned to a single computational object. This means that this object is entirely responsible for the fulfillment of the composition constraints

A typical framework that can be used with the first implementation pattern is Subject-Oriented Programming (SOP) [Harrison93]. The well-known SOP tool is Hyper/J [Tarr00] that allows a developer to specify roles as separate source code files and then compose them based on a special option file. This option file indicates how source code files participate in a composition: how equal named parameters or actions should be treated, and other complimentary information [Tarr00]. The result of the composition is a new source code file that that can be compiled in a usual way.

Using SOP, the implementation of roles with the first implementation pattern will require the following steps:

- Generation of code for each role in a model;
- Generation of option files from the composition constraints;
- Composition of roles' codes into a new source code using the Hyper/J tool, compilation and the execution of the new source code.

Similarly to SOP, Aspect-Oriented Programming (AOP) [Kiczales97] can be used with the first implementation pattern. In the Application to Software Engineering section of Part 3 we show more details about SOP and AOP as tools for the implementation of role model compositions.

Let's consider the main benefits and limitations of the first implementation pattern:

Benefits:

- Reliable: the sate of the composed role is controlled by a single object.

Limitations:

- Does not allow for the dynamic role reconfiguration: roles are composed statically. Any reconfiguration (changes in composition constraints) will require the regeneration of a composed role.
- Does not allow a modeler to separate base roles to implement them with different computational objects (to keep them separate, for example for the purpose of distribution).

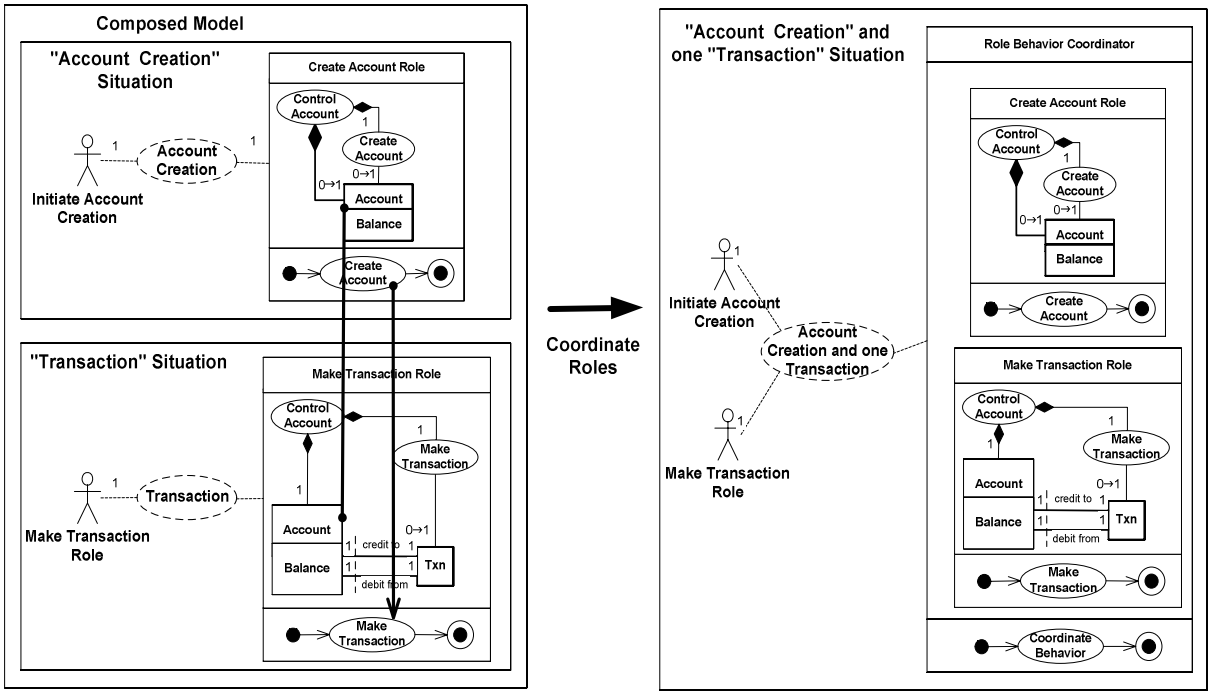


Figure 57. Implementation patterns:
Role Model Composition Specification and its implementation: Coordinating Base Roles

The second implementation pattern, "*Coordinating Base Roles*", keeps the specifications of base roles unchanged. It "implements" composition constraints by means of coordinating the behavior of base roles with an additional behavior. In Figure 57 we represent this behavior as the "Role Behavior Coordinator" role that incorporates two base roles. In this example the "Role Behavior Coordinator" guaranties the equality of attributes and the sequentiality of "Create Account" and "Make Transaction" actions.

A typical framework that can be used with the first implementation pattern is Coordination Development Environment (CDE) [Andrade99], [Fiadeiro97]. With this framework roles are implemented as different Java objects. Composition constraints are implemented as a coordination contract (we call it "coordinate behavior" action in Figure 57). First, Java objects are deployed in CDE, then a coordination contract is deployed in CDE. Once a coordination contract is deployed, CDE intercepts all messages intended for base roles and then follows the contract. For each intercepted message a contract may specify an additional behavior; it can block or enable the original behaviors of the base roles.

In a similar way the Sina programming language [Aksit88] is built. It includes *composition filters* that are used to intercept and redirect messages.

Let's consider the main benefits and limitations of the second implementation pattern:

Benefits:

- Allows for the dynamic reconfiguration: composition constraints can be reconfigured in runtime by replacing the existing coordinated behavior with a new one;
- Base roles can be distributed between two objects, which allows for the distribution of responsibilities between objects.

Limitations:

- May require a synchronization or complex communications between objects that play base roles in order to guarantee composition constraints. Therefore, solutions based on this implementation pattern may suffer from efficiency and scalability problems.

APPENDIX 2

DIFFERENT MEANINGS ON THE TERM ROLE

The notion of role that is central for our Situation-Based Modeling Framework. Let's note that the meaning of the term role can be understood differently depending on the background of our readers. In this appendix we consider the two meaning of the term role: role as "an object or a behavior" (Section 1) and role as named place in a relationship (Section 2).

1. Role: an object or a behavior?

One of the major differences in the definition of the term role is related to the perception of roles as behaviors or objects (better to say "role objects"):

- Behavior: Role as a behavior is usually used in the phase of system requirements specifications. It represents a functionality of a system limited to a certain context.
- Role object: Role as an object is usually considered in the implementation phase. "Intuitively, the idea of role as an object type makes perfect sense. It is indeed easy to conceive of a collection of objects that are susceptible to perform a particular role. Readers familiar with Java or UML have already encountered a similar idea, as interface in both this languages is a type concept applicable to objects" [Genilloud00].

The classification of roles as behaviors or "role objects" is related to the dynamic/static notion of generalization. To see this difference we refer to the following examples, taken from F. Steimann's paper [Steimann00] (see Figure 58).

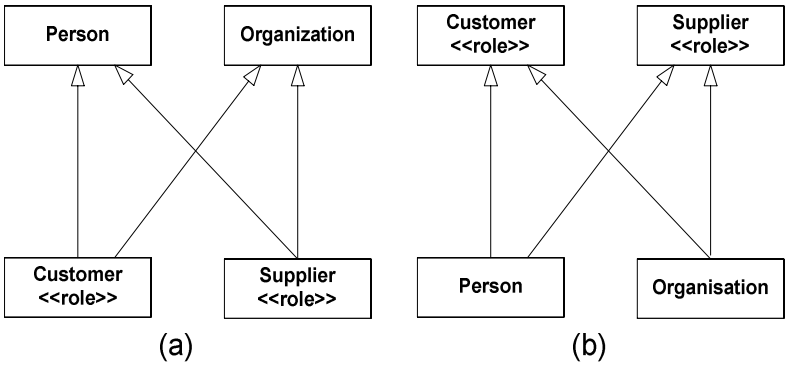


Figure 58. Roles as behaviors versus roles as objects.

Steimann in [Steimann00] explains that roles can be regarded as specializations or generalizations of "natural" types[28]. Figure 58 represents roles as:

- Specializations of natural types (Figure 58.a): If we consider Person as a "natural" type object, than Customer and Supplier would be its subtypes. These two subtypes can be defined as roles of a person. In the same way an organization may have the same two roles.
- Generalizations of natural types (Figure 58.b): We can consider Customer as a supertype of the Person and Organization natural types. This supertype can be defined as a role[29] that can be played by both the

[28] Sowa in his monograph on conceptual structures [Sowa84] considers natural types as characteristics "that relate to the essence of the entities".

Customer or Person objects. In this case customer represents a kind of interface that can be bound to any object with a compatible type (such as Customer or Person).

The result of these two specializations/generalizations (seen in Figure 58) of "natural" types is a paradox that a role can be defined as both: subtype and supertype of an object type. Steimann in [Steimann00] explains this paradox with the fact that in the first case roles are defined as *dynamic* subtypes and in the second case roles are defined as *static* supertypes. The dynamic subtyping means that the set of persons (in Figure 58.a) is dynamically divided into the set of customers and suppliers. A person can dynamically change its roles, i.e. a person may abandon the role of Customer and start playing the role of Supplier. The static supertyping means that a set of customers (in Figure 58.b) is statically partitioned into the set of persons and organizations. Customer is either Person or Organization (not both of them).

The aforementioned paradox is related to our question in the beginning of this subsection, if a role is an object or a behavior. The relation is the following: roles as behaviors can be seen as dynamic specializations of natural types; roles as objects types can be seen as static generalizations of natural types.

The solution to our question and the aforementioned paradox is that roles can be understood both ways: as behavior and objects. However these two notions of roles should be separated. "The solution, clearly, lies in the separation of types [i.e. roles as objects] and roles [i.e. roles as behaviors]. If the type and role hierarchy are different hierarchies, none of the problems related to subtyping occurs" [Steimann00].

In the context of EA the separation of two aforementioned hierarchies can be done in a straightforward way, by means of introducing organizational and functional levels hierarchies. "The functional levels represent the behavioral hierarchy whereas the organizational levels represent constructional hierarchy" [Rychkova03], [Wegmann04]. The notion of roles as behaviors is related to the behavioral hierarchy (see Figure 59).

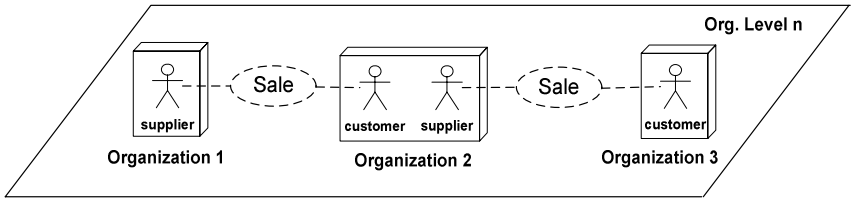


Figure 59. Roles as behaviors versus roles as objects.

In Figure 59 the customer and supplier roles represent the behaviors of organizations (Organizations 1, 2 and 3) in the Sell collaborations. Note that these behaviors can be further refined into more detailed behaviors (or more specialized roles). This refinement of roles results in the functional hierarchy of roles. In our framework the term role is understood based on the perception of roles as behaviors: role defines the partial behavior of an object in a relation to a certain situation.

The notion of roles as objects is related to the organizational hierarchy (see Figure 60).

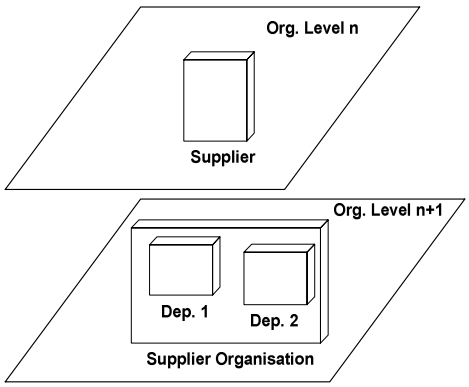


Figure 60. Roles as behaviors versus roles as objects.

In Figure 60 the "Supplier" object at the Org. Level n is modeled as the "Supplier Organization" with two internal departments (Dep. 1 and 2) at the Organizational level n+1. In this example Supplier can be considered as a role of these to departments in an organization. Two departments, internal for the "Supplier Organization", can be further refined into smaller objects. This refinement results in the organizational hierarchy of objects. In

[29] This notion of role is often referred as the named place in a relationship. In UML, for example, roles are represented as 'labels' linked by to a relationship.

our framework the notion of role as an object (such as a supplier in Figure 60) corresponds to changing between organizational levels: we specify an object (something similar with interface) at one organization level and its actual implementation at the next organization level.

2. Role as a Named Place in a Relationship

Another common notion of a role is the named place in a relationship. This notion of a role exists in many notations, such as ER, UML, and IDEF.

We already mentioned above that in our framework we distinguish between computational and information viewpoints. Therefore, we have to see how a named place in a relationship can be understood from both these viewpoints.

Computational Viewpoint (CV)

In CV, objects use roles identifiers to get access to features (data and processes) of other objects. Based on this idea, a role is defined in UML: "A role specifies (through its type) the required set of features a participating instance must have" [OMG01]. For example, in Figure 61 the r1 role specifies the set of features (r1 related data and r1 related behavior) of the Object 2 that the Object 1 is aware of. These features are known by the Object 1 through the identifier of the role. The availability of these features assumes the existence of collaboration between objects (see Figure 61.b). As we explain later, a collaboration has its own life cycle (collaboration has to be created, maintained (persisted) and deleted). In general case both objects may include behavior related to the support of the collaboration life cycle. Therefore, in a CV a role (Figure 61.a) refers to the behavior and data encapsulated in both objects connected by the relationship (Figure 61.b).

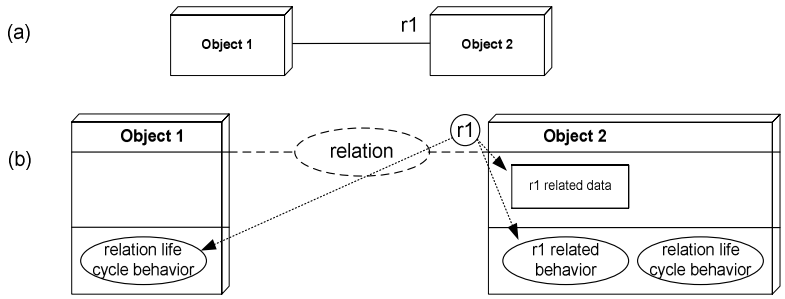


Figure 61. Role as a Named Place in Relationship: (a) abstract form, (b) detailed form

Note that in our framework we show explicitly the collaborations between objects rather than simply relations (an abstract form of collaborations). We also show explicitly all roles that participate in collaborations (contrary with the example from Figure 61 where only one role (r1) is shown).

Information Viewpoint (IV)

IV is based on conceptual models that represent concepts, known by a system, and relations between these concepts. Based on Sowa, relations between concepts and roles as named places in relationships can be considered as predicates. "In predicate calculus, roles are usually represented by dyadic predicates that relate a prehending entity to a prehended entity" [Sowa99]. In a visual form roles are usually represented as holes in some form of visual predicates. For example, in Object Role Modeling (ORM) [Halpin01] a predicate is just a sentence with object holes in it (see Figure 62.b). The ovals in ORM diagrams represent object types. These are connected to predicates, shown as sequences of boxes (or holes). Each box corresponds to a role in the relationship. Such a way of representing relations as predicates is a strong point the ORM method. This is especially true for n-ary relations. In Figure 62.b, for example, you can easily read "X works for Y" by filling the corresponding holes in the ORM predicate. However, in other languages (such as UML class diagram, Figure 62.a) mapping relations to predicates is less intuitive (especially true for n-ary relations). Role names in UML class diagrams correspond to holes in predicates. These roles are occupied by objects instances of the corresponding object types (X and Y).

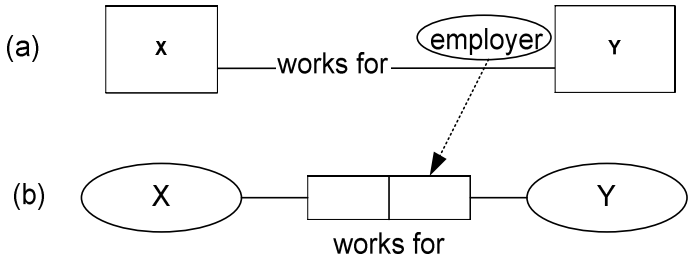


Figure 62. Role as a Named Place in Relationship: (a) in UML (b) in ORM

In Section 4 (Axiomatic Semantics of our Modeling Language) of this part we explain how the conceptual models in our framework can be mapped to the Alloy language based on predicate logic. We use this mapping to check the correctness of our conceptual models.

APPENDIX 3

FROM RM-ODP TO THE FORMAL BEHAVIOR REPRESENTATION

Overview. In this appendix we consider the behavioral aspects of system modeling. In order to specify the behavior of a system, many different notations can be used. Quite often, different terms in these notations are related to the same element in a system implementation. In order to relate these terms and guarantee the consistency between different notations, a standard framework should be used. In this work we show how the Reference Model for Open Distributed Processing (RM-ODP) can be used for the purpose of the mapping of terms from different behavioral notations. RM-ODP behavior models are based on the concept of Time Specific Action. Time Specific Actions represent directly things that happen in the Universe of Discourse with explicit reference to time. However the explicit reference to time leads to a considerable loss of abstractness. To elevate the level of abstraction we have considered Time Abstracted RM-ODP models where concrete time information is omitted. We used Time Abstracted RM-ODP models to show the correspondence between terms in UML Activity Diagrams, UML Statechart Diagrams and CCS process algebra by means of relating them with RM-ODP terms. This allows us to consider RM-ODP as a possible meta-model for behavior specifications written in UML. It can help to insure the consistency of UML models.

1 Introduction

Behavior models play a central role in system specifications. Many specification languages can be used to specify the behavior of a business and IT systems. A system designer chooses a particular language depending on the designer's experience and on the problems he is trying to solve. For example, to show the conformance of the implementation of a system behavior with its specification, a system designer can use formal languages (for example, Pi-calculus). To visualize the state machine of a developed system, a system designer may use a UML statechart diagram or activity diagram (a variation of a state machine in which the states represent the performance of actions or subactivities [OMG01]). The design of complex systems requires that a system designer solve many problems simultaneously (visualize a model, check the conformance of a model, etcetera), thus several specification languages should be used. This raises a problem: a system designer needs to build several independent models of the same system. This leads to the duplication of the information, which can be an additional source of errors: models done in different languages can be inconsistent.

To avoid building several mutually dependent models, we can build a generic model (see Figure 63).

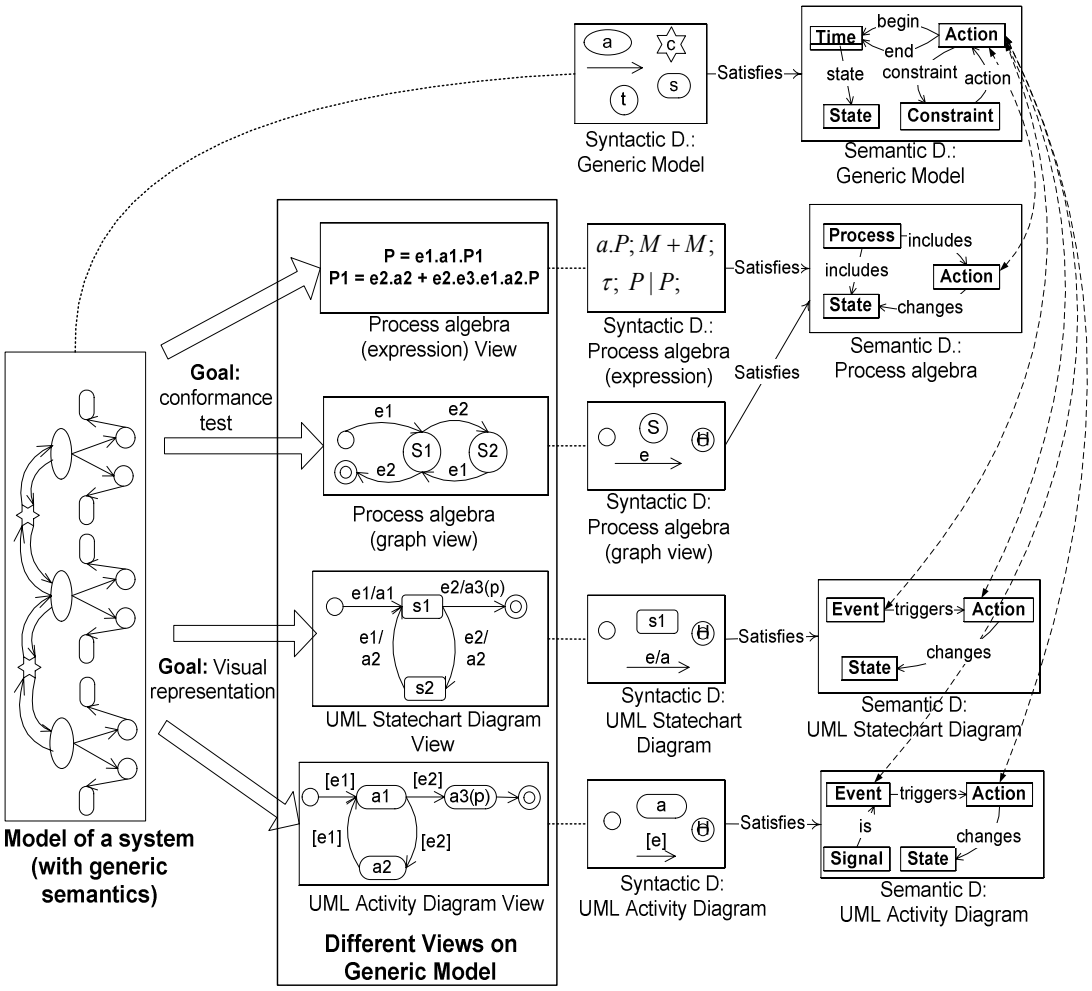


Figure 63. Generic Model and different views on a generic model

All other models can be considered as views of this generic model. Any view on a model should address some particular problems that a system designer wants to solve. Any view is based on a particular specification language. J. Wing in [Wing90] defines specification language as a triple <*Syntax, Semantics, Satisfies*>, where *Syntax* is called a language syntactic domain; *Semantics* is a language semantic domain, and *Satisfies* is "satisfies relation". *Satisfies relation* defines the relation between syntactic terms in a system specification and their semantic meanings in a semantic domain.

The semantic domain of the generic model should cover the semantics of all possible views that a generic model can have. In other words, any concept in the semantic domain of any view should be mapped with one or more concepts in the semantic domain of the generic model. This shows that the semantic domain of the generic model should be generic enough to include all fundamental concepts for the specification of business and IT systems.

In this work we propose to use the Part 2: Foundations of the Open Distributed Processing - Reference Model RM-ODP [ISO96] as a semantic domain for the generic model. "RM-ODP, ITU-T Recommendations X.901 to X.904 | ISO/IEC 10746, is based on precise concepts derived from current distributed processing developments" [ISO96]. We choose RM-ODP because "RM-ODP introduces generic terms that apply to any form of modeling activity" [Naumenko01] and RM-ODP provides rigorous semantics for these terms. We use a formalization of this semantics written in the Alloy language[30]. This formalization was proposed in [Naumenko01], where Naumenko shows the classification of RM-ODP concepts with the aid of the set theory and using regular predicate logic. RM-ODP concepts described in [Naumenko01] can be used only as a basis for the semantics of the generic behavior model. The work of Naumenko contains too many different concepts: not all of them are related with behavior modeling. Thus in this work we consider only a subset of concepts from [N2000] and refine some of them in order to define semantics for generic behavior models.

[30] "Alloy is a language for describing structural properties. It offers declaration syntax compatible with graphical object models, and a set-based formula syntax powerful enough to express complex constraints" [Jackson00]. See also http://sdg.lcs.mit.edu/alloy/.

In section 2 we consider the minimum set of RM-ODP concepts that we need to build generic behavior models. We define more precisely some RM-ODP behavior modeling concepts (particularly behavioral constraints, time and state). We have to do this because RM-ODP does not define all modeling concepts precisely enough to relate them with other existing formal notations. One of the basic RM-ODP modeling concepts that we consider in section 2 is action. It represents directly things that happen in the Universe of Discourse with explicit reference to time. In other words, any action (or time specific action) is specified for the particular time interval. We call a model built with time specific actions Time Specific RM-ODP model.

Time Specific RM-ODP model can not be used to specify an infinite behavior that may contain infinitely many actions. To specify infinite behavior, a system designer has to use action types. In Section 3 we introduce two action types: Time Abstracted Action (Section 3.1) and Parameterized Time Abstracted Action (Section 3.1). Based on these two types, we define a Time Abstracted RM-ODP model. This model can be taken as a generic from Figure 63. The main contribution of the work related this appendix consists in making explicit the relations between Time Specific and Time Independent RM-ODP models.

In section 4 we show an example of how a Time Abstracted RM-ODP model can be used as a generic model. We show how it can be seen from the three views done with the following specification languages: CCS process algebra, UML Activity Diagram and UML Statechart Diagram. Section 5 is a conclusion.

2 RM-ODP a Generic Semantic Domain

In this section we consider the concepts from the RM-ODP semantic domain that are necessary for the modeling of the behavior of systems.

The basic concepts that we use in our work are taken from the clause 8 "Basic modeling concepts" of the RM-ODP Part 2. These concepts are: action, time, and state. According to [Naumenko01] these concepts are essentially the first-order propositions about model elements. We will also use some concepts (type, instance, precondition, postcondition) from the clause 9 "Specification concepts". Specification concepts are the higher-order propositions applied to the first-order propositions about the model elements. Wegmann [Wegmann01] states: **"***Basic Modeling Concepts* and generic *Specification Concepts* are defined by RM-ODP as two independent conceptual categories. Essentially, they are two qualitative dimensions that are necessary for defining model elements that correspond to entities from the universe of discourse".

To explain the semantics of the generic model more clearly, we will use the Alloy formalism. Alloy is a simple modeling language that allows a modeler to describe the conceptual space of a problem domain. Using Alloy we specify the RM-ODP semantic domain.

RM-ODP conceptual elements from the semantic domain can be partitioned in the following way:

```
model RM-ODP {
domain {ODP_Concepts}
state {
partition … BasicModellingConcepts, SpecificationConcepts : static ODP_Concepts
…
}
```

Code Fragment 1. RM-ODP model

Let's consider the minimum set of modeling concepts (Basic Modeling Concepts and Specification Concepts) necessary for the specification of systems behavior. There are a number of approaches for specifying the behavior of distributed systems coming from people with different backgrounds and considering different aspects of behavior. "However, they can almost all be described in terms of a single formal model" [Lamport90]. Based on Lamport, to specify the behavior of a concurrent system a system designer has "to specify a set of states, a set of action and a set of behavior". Each behavior is modeled as a finite or infinite sequence of interchangeable states and actions. To describe this sequence there are mainly two dual approaches. According to [Broy91] they are:

1. "Modeling systems by describing their set of actions and their behaviors".
2. "Modeling systems by describing their state spaces and their possible sequences of state changes".

"These views are dual in the sense that an action can be understood to define state changes, and state changes occurring in state sequences can be understood as abstract representations of actions" [Broy91]. In our work we consider both of these approaches as an abstraction of the more general approach based on RM-ODP. In the next subsection we consider the first approach where we give the definition of action and behavior. Then we consider the definition of state and state structure. Finally we show how state and behavior are related, thus showing their duality.

2.1 Action Structure

In this subsection we show how systems are specified "by describing their set of actions and their behaviors". Action in RM-ODP is defined as:

Action: "*Something which happens"*.

This definition means that "action characterizes a model element for its being "something that happens" [Wegmann01]. To specify a model element as an action we have to consider two other modeling concepts that model changes happening in a system when an action occur. They are *state* and *time*. The definition of the state concept is given in the next subsection. The concept of time is a fundamental concept in modeling of systems. Based on RM-ODP time is a basic modeling concept that is used to specify the beginning and the end of an action[31]. Therefore each RM-ODP action is bound to the specific time interval. That is why in our work we call RM-ODP action as *Time Specific Action (TSAction):*

```
partition …, TSAction, Time, … : static BasicModellingConcepts
                                 // Time and TSAction are BasicModellingConcepts

instant_begin : TSAction –>Time!        // each TSAction has one time point when it starts
    instant_end : TSAction –>Time!      // each TSAction has one time point when it finishes
```

Code Fragment 2. Beginning and end of TSAction

However RM-ODP does not explain how time is modeled. A system designer has to decide how accurate he wants to model time. Henri Poincaré in [Poincaré83] shows that a precise clock that can be used for time measurement does not exist in practice but only in theory. So the measurement of the time is always approximate. In this case we should not choose the most precise clocks, but those that explain the investigated phenomena in the best way. "Simultaneity of two events or their sequentiality, equality of two durations should be defined in the way that the formulation of the physical laws is the easiest" [Poincaré83]. According to this idea we can choose different models of time. RM-ODP confirms this idea by saying that "a location in space or time is defined relative to some suitable coordinate system" [clause 8.10]. The time coordinate system defines a clock used for system modeling.

In our work we consider a time coordinate system as a partially ordered set of time points. Each point can be used to specify the beginning or the end of TSAction. A time coordinate system must have the following fundamental properties:

- Time is always increasing. This means that sequences of time points can not have loops.
- Any time point is defined in relation to other time points (next, previous or not related). This corresponds to the partial order defined on the set of time points.

We use the following formalization of time in Alloy: time is defined as a set of time points. Any time point has to be defined in relation with some other time points (partial order):

```
nextTE: Time -> Time        // defines the set of nearest following time points for any time point
                            // note that any time point may include several nextTE time points
```

We will also use the followingTE relation to define the set of the following time points or the transitive closure of the time point t over the nextTE relation:

```
// part of Alloy time declaration
followingTE: Time ->Time      // defines all possible following time points
```

Using followingTE we can write the following Alloy invariant[32] that defines the transitive closure and guarantees that time point sequences do not have loops:

```
inv TimeInvariant {
all t: Time  |                                          // For all time points t
((no t.nextTE)->(no t.followingTE)) &&                  // (if t does not have nextTE it also does not have followingTE) and
 ((some t.nextTE && no t.nextTE.followingTE)            // (if t has the nextTE that does not have any followingTE
   ->(t.followingTE=t.nextTE )) &&                      //  then t.followingTE is equal to t.nextTE) and
   ((some t.nextTE && some t.nextTE.followingTE)        //  (if t has the nextTE that has some followingTE
   ->(t.followingTE=t.nextTE.followingTE + t.nextTE )) &&  //   then t.followingTE includes t.nextTE and t.nextTE.followingTE) and
(t not in t.followingTE)                                // (time does not have loops)
}
```

Code Fragment 3. Time invariant

[31] "Location in time: An interval of arbitrary size in time at which action can occur." [I1996]

[32] Do not confuse this Alloy invariant with the invariant defined in RM-ODP. We use an Alloy invariant to guaranty the consistency of concepts on the meta-level. This invariant can not become a part of our model, while the RM-ODP invariant is a specification concept. It is a predicate that can be used in a model. In this work we use the concept of invariant only at the meta-level.

Now, using the already defined concept of *Time* we can give a formal Alloy definition of *TSAction*:

```
def TSAction{
  all a: TSAction                    // for each TSAction a
  | some t1:a.instant_begin          // [ (exists t1 = a.instant_begin) and
  | some t2: a.instant_end           // (exists t2 = a.instant_end) ] then
  | (t2 in t1.followingTE)           // (t2 happens after t1)
}
```

Code Fragment 4. TSAction

In this definition we suppose that the duration of any TSAction is not equal to zero (t1 can not be equal to t2). But in certain cases we can make an abstraction of the information about the fact that TSAction starts and ends in different time points (to define so called instantaneous actions). For this purpose we have to use an abstraction of time information that we consider in section 3.

To make a specification that includes more than one TSAction, we have to consider how TSActions in a specification can be structured. We use the RM-ODP *behavior* concept to define the TSAction structure:

Behavior: "*A collection of [Time Specific] Actions with a set of [Time Specific Behavioral] Constraints on when they may occur",*

That can be formally represented in the following way:

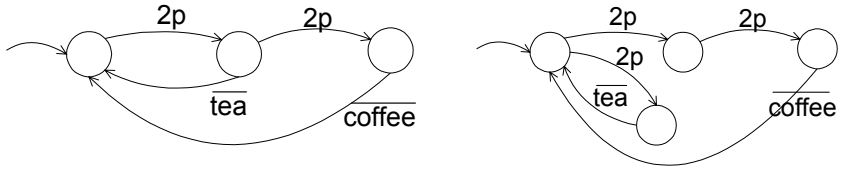
```
// part of Alloy behavior declaration
Behavior: BasicModellingConcepts
partition TSAction, TSBehavioralConstraints: static Behavior
        // Behavior is partitioned into the set of actions and the set of constraints.
corresponding_constraint (~constrained_action) : TSAction -> TSBehavioralConstraints
        // TSActions defined with corresponding TSBehavioralConstraints and vice versa.

def Behavior {
all b: Behavior |          // For any element b from Behavior set; (note that behavior is
                           // partitioned into the set of TSActions and the set of
                           // TSBehavioralConstraints)
  ( (b in TSAction) &&                    // [ (if b is a TSAction) then
  (some b.corresponding_constraint) )) ||  //   (b has a at least one corresponding_constraint) ] and
  ( (b in TSBehavioralConstraints) &&     // [ (if b is a TSBehavioralConstraint) then
  (some b.constrained_action) )           //   (b has a at least one constrained_action) ]
}
```

Code Fragment 5. Behavior

This definition uses a concept called *(TimeSpecificBehavioral) constraint*. RM-ODP does not give us the precise definition of these constraints. But it gives some examples. Constraints may include, for example, constraints of sequentiality, non-determinism, concurrency or real-time constraints. From the definition of behavior, we can only conclude that TSBehavioralConstraints are part of a system behavior and that they are associated with TSActions (see the formal definition above). We will extend the definition of behavioral constraints in the next subsection.

2.1.1 Time Specific Behavioral Constraints

Many modeling techniques represent behavioral constraints implicitly. Quite often we can infer them from behavior representation, like a transition graph. For example, Figure 64 shows an example from the Milner's book [Milner99] with two different specifications of a coffee/tea vending machine. This machine accepts coins of value 2p and provides a customer with coffee or tea. To get a coffee or tea a customer has to introduce coins and press a corresponding button (coffee or tea). The price for tea is 2p and the price for coffee is 4p. Figure 64.a shows the specification that has only constraints of sequentiality, since in any state of a system the next action is precisely defined depending on the request of a customer. Figure 64.b shows the specification with constraints of sequentiality as well as constraints of non-determinism. We can infer that the system in Figure 64.b is specified using constraints of non-determinism; "after we have put in the first 2p, it may be in a state in which we can only get tea (it will not accept a further 2p), or it may be in a state in which we can only put in more money to get coffee" [Milner99]. These two specifications "are annoyingly different for a thirsty user"



(a) **(b)**

Figure 64. Specification of the system using: a - sequential deterministic constraints; b - sequential and non-deterministic constraints

We base our approach on RM-ODP, where BehavioralConstraints are represented explicitly ("Behaviour of an object: A collection of (TS) Actions with a set of (TS) Behavioral Constraints on when they may occur" [ISO96]). In our work we show how TSBehavioralConstraints can be made explicit: how the behavior of a system can be specified using a set of TSAction and TSBehavioralConstraints of sequentiality and non-determinism.

2.1.2 Constraints of Sequentiality

We begin with the analysis of TSBehavioralConstraints of sequentiality (TSSeqConstraints in Alloy code fragment 6). Each TSSeqConstraint of sequentiality should have the following properties:

- It is defined between two or more TSActions.
- Sequentiality has to guarantee that one TSAction is finished before the next one begins.

```
TSSeqConstraints: TSBehavioralConstraints                    // TSSeqConstraints are TSBehavioralConstraints

def TSSeqConstraints {
all sc: TSSeqConstraints |                                   // for any sc: TSSeqConstraints
some a1, a2: TSAction | (a1 != a2) &&                        // (there are two different TSActions a1, a2) such that
    (a1 in sc.constrained_action) && (a2 in sc.constrained_action) &&  // (sc is defined between a1 and a2) and
   ( (a2.instant_begin in a1.instant_end.followingTE) ||     // [ (a1 is before a2) or
   (a1.instant_begin in a2.instant_end.followingTE) )        //   (a2 is before a1) ]
}
```

Code Fragment 6. TSBehavioralConstraints of Sequentiality

The Alloy definition from the code fragment 6 requires TSSeqConstraints to have a minimum of two sequential actions that happen one after another. But this Alloy definition does not tell us which TSActions happen first. To specify this we use two Alloy relations (seq_constraint and next_actions) and SeqInvariant (see code fragment 7). The seq_constraint relation relates a given TSAction (let's call it *tsa)* to one TSSeqConstraint. Then the next_actions relation relates TSSeqConstraint to the set of the TSActions. This set of action is the set of next TSActions for *tsa.*

```
seq_constraint: TSAction->TSSeqConstraints!  // for any TSAction there is one TSSeqConstraint that connect TSAction with
                                             next TSActions
next_actions: TSSeqConstraints -> TSAction   // any TSSeqConstraint can have several next TSActions
inv SeqInvariant {
  all sc:SeqConstraints |                    // for any sequential constraints sc and
     all a1:sc.constrained_action |          // for all TSActions a1 and a2
     all a2:sc.constrained_action |          // constrained by sc

    (  (a2.instant_begin in
            a1.instant_end.followingTE) ->   //  if a1 is before a2 then
              ( (sc=a1.seq_constraint) &&    // [ (sc is seq_constraint for a1) &&
                (a2 in sc.next_actions) &&   //   (sc includes a2 as the next action) &&
                (a1 not in sc.next_actions) &&  //  (sc does not include a1 as the next action) &&
                (sc not in a2.seq_constraint)  ) //  (sc is not sequential constraint for a2) ]
     ) &&                                    //  AND
     (  (a1.instant_begin in
              a2.instant_end.followingTE) ->  //  if a2 is before a1 then
                ( (sc=a2.seq_constraint) &&  // [ (sc is seq_constraint for a2) &&
                  (a1 in sc.next_actions)  &&  //  ((sc includes a1 as the next action) &&
                  (a2 not in sc.next_actions) &&  //  (sc does not include a2 as the next action) &&
                  (sc not in a1.seq_constraint)  ) //  (sc is not sequential constraint for a1) ]

    )
}
```

Code Fragment 7. Invariant that defines the sequence of TSActions

To illustrate the Alloy definition of TSSeqConstraints we show the example of the model (see Figure 65) that corresponds to the formal Alloy semantics given above.

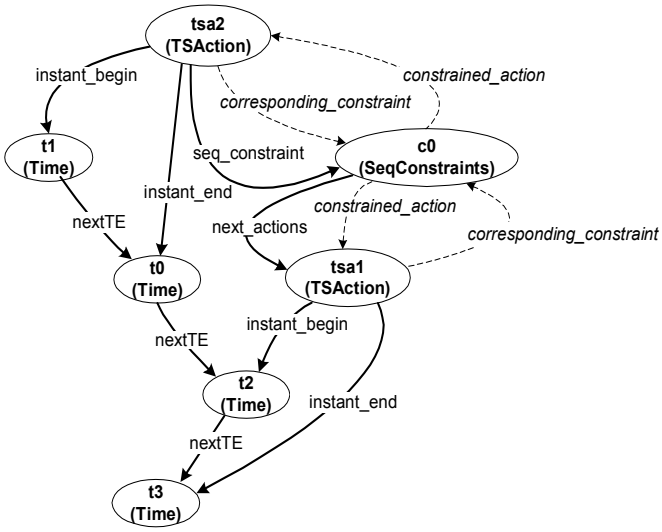


Figure 65. Example of the model of a system behavior built with Alloy Constraint Analyzer

The model from Figure 65 was built with the Alloy Constraint Analyzer[33]. This model is a result of the analysis of formal behavior semantics done with alloy Constraint Analyzer. The Alloy Constraint Analyzer checks the consistency of the formal semantics, randomly generates a sample configuration and visualizes it. Figure 65 shows a model that consists of the set of TSActions {tsa1, tsa2}, the set of TSBehavioralConstraints {c0}, the set of time points {t1, t0, t2, t3} and relations between model elements. Note that labels for model elements in Figure 65 are generated automatically. That is why these labels are not ordered. We can see that the constraint c0 is the TSBehavioralConstraint of sequentiality between two TSActions tsa1 and tsa2. In Figure 65 we show the corresponding_constraint and constrained_action relations with dotted arrows. We do it because these two relations do not do not have a particular interest for us for the rest of this work. They have been used only to define TS behavioral constraints. Thus we do not show these relations in following figures. Instead we use the seq_constraint and next_actions relations to show the sequence of TSActions.

The fact that the Alloy Constraint Analyzer has found a sample model allows us to conclude that formal behavioral semantics done in Alloy does not contain contradictions.

The definition of the constraints of sequentiality allows us to specify the semantics of the concepts defined in the section 13 of RM-ODP "Activity[34] Structure". Here we give two examples (for Chain of actions and Head action) that show how the formal semantics for these two concepts can be done based on the constraints of sequentiality.

Head action: *In a given activity, an action that has no predecessor.*

```
def HeadAction{
  all ha:HeadAction|                          // for all ha:HeadAction
       no a:TSAction|                         // does not exist any a:TSAction
       ha in a.seq_constraint.next_actions    // such that ha is successor of a
}
```

Additionally we have to guarantee that all TSActions that do not have predeccessors are Head actions. We do it with the following Alloy invariant:

```
inv HeadActionInvariant {
  all a:TSAction| (no a1:TSAction| a in a1.seq_constraint.next_actions) ->(a in HeadAction)
}
```

Another concept that can be formalized using constraints of sequentiality is a chain of actions:

Chain (of actions): *A sequence of actions within an activity where, for each adjacent pair of actions, occurrence of the first action is necessary for the occurrence of the second action.*

Based on the definition of composition constraints, we have to require that for any action in a chain maximum one successor and maximum one predecessor is possible:

```
def Chain {
```

[33] See http://sdg.lcs.mit.edu/alloy/

[34] RM-ODP defines activity in the following way: "**Activity:** A single-headed directed acyclic graph of actions, where occurrence of each action in the graph is made possible by the occurrence of all immediately preceding actions."

```
all ch:Chain |                                          // for all chains of actions
    (not sole ch.actions_in_chain ) &&                  // {there are min 2 action} &&
    ( all a:ch.actions_in_chain |                       // {for all actions a in chain ch:
     (   ( one a1: ch.actions_in_chain |                //   [ (there is one
          a in a1.seq_constraint.next_actions) ||       //     predecessor action a1) or
        ( a in HeadAction )  ) &&                       //     (a is Head action) ] &&
    ( one a.seq_constraint.next_actions ||              //   [ (there is one successor) or
      no a.seq_constraint.next_actions ) &&             //     (there is no successors) ] &&
    ( one a2: ch.actions_in_chain | a2 in HeadAction )  //   [one Head action per chain]}
    )
}
```

2.1.3 Constraints of Non-determinism

In order to formalize TSBehavioralConstraints of non-determinism we considered the following definition given in [Broy91]: "A system is called non-deterministic if it is likely to have shown a number of different behaviors, where the choice of the behavior cannot be influenced by its environment". This definition of non-deterministic constraints is given from the point of view of the external observer of a system: when the external observer can not predict the reaction of a system after an interaction with a system. This means that the system at one point makes an internal choice between a minimum of two "branches" of different behavior.

Let's see how this definition works for the example from Figure 64.b. In Figure 64.b we can see that when a user of the coffee machine introduces first *2p*, the system can enter into two different states and therefore it can have two different behaviors: it will wait for the second *2p* or will provide *tea* for the user of the coffee machine. Thus a system has two different behaviors and the choice of the behavior can not be influenced by its environment.

In a general form, TSBehavioralConstraints of non-determinism should be defined between a minimum of three TSActions. The first TSAction should precede the two following internal TSActions. We can write this in Alloy in the following way:

```
TSNonDetermConstraints: TSBehavioralConstraints     // TSSeqConstraints are TSBehavioralConstraints
def TSNonDetermConstraints {
all ndc: TSNonDetermConstraints |                   // for any ndc: TSNonDetermConstraints
   some a1:TSAction |                               // (there is an TSAction a1) and
   some a2, a3 in  InternalTSAction |               // (there are two internal TSActions a2 and a3) such that
   (a1 in ndc.constrained_action) &&                // (sc is defined for a1) and
   (a2 in ndc.constrained_action) &&                // (sc is defined for a2) and
   (a3 in ndc.constrained_action) &&                // (sc is defined for a3) and
   (a2.instant_begin in a1.instant_end.followingTE) &&  // (a1 is before a2) and
   (a3.instant_begin in a1.instant_end.followingTE)     // (a1 is before a3)
}
```

Code Fragment 8. Constraints of non-determinism

Note that intuitively we may think to model a constraint of non-determinism as an internal action that makes a non-deterministic choice between two (or more) following actions. Can we really do that? An action that makes a choice between two "branches" of behavior should be specified with two (or more) different post-states.[35] Each post-state defines a separate "branch" of behavior. But in our case we use time specific actions. This means that each action has a particular time when it starts and ends. As we will show in the next section, each time moment is associated to only one state. Thus the specification of a non-deterministic choice is not possible using TSAction and we use behavioral constraints to represent it in our models.

The discussion from the previous paragraph shows that the semantics of behavioral concepts would not be complete without considering the state of an object: "an object is characterized by its behavior and, dually, by its state" [ISO96]. In the next section we discuss the definition of the state of an object and relate the concept of state with behavioral concepts considered above.

2.2 State Structure

Here we consider the second approach based on "Modeling systems by describing their state spaces and their possible sequences of state changes" [Broy91]. We begin with RM-ODP definition of *state*:

[TS]State (of an object) (RM-ODP, Part 2, clause 8.7): *At a given instant in time, the condition of an object that determines the set of all sequences of [TS]Actions in which the object can take part.*

[35] Post-state is a state of an object after the occurrence of an action.

This definition shows that the state of an object is defined in a given time point. That is why we call this state as Time Specific State (TSState).

In this work we use some simplifications. Since in our work we consider the behavior only for one object, we do not make objects explicit on diagrams and in Alloy code. Therefore we declare TSState in Alloy without making a reference to an object:

```
// part of Alloy state declaration
state-existence: Time! -> TSState_!          // state is defined at a given moment in time
```

This Alloy definition taken from [Naumenko01] can hardly be used in practice: to make specifications of complex systems it is not enough to specify TSState of an object in any point in time. We have to specify particular details that show how the TSState of an object changes. For this purpose we use the state structure:

TSState Structure (of an object): *A set of attributes, a set of attribute values.*

Based on the TSState Structure we can specify states of each attribute. The state of an attribute specify the value that this attribute has in a given time point. Each action can change values of some attributes while keeping other attributes unchanged. The composition of states of all attributes of an object gives us the composite state:

Composite TSState (of an object)*: Composition of states of all attributes of an object.*

To specify the Composite TSState of an object we will use a function that specifies the relation between attributes and their values at a given moment in time. In Alloy this definition is written in the following way:

```
// part of Alloy declarations
partition … Information … : static BasicModellingConcepts      // Information is a basic modeling concept
partition StructuralInfo, BehavioralInfo : static Information      // Information can be structural and behavioral
// State Structure
Attrs, AVals: StructuralInfo            // state structure: set of attributes and attribute values
    attrValue [Time]: Attrs -> AVals!     // any attribute has one value at a given moment
```

Code Fragment 9. Structural and behavioral information

Note that our definition of a Composite TSState extends the definition of the state proposed in RM-ODP. A Composite TSState shows how RM-ODP state can be specified as a composition of the states of several attributes.

As we said above "an action can be understood to define state changes and state changes occurring in state sequences can be understood as abstract representations of actions" [Broy91]. This shows that TSState is dual with the concept of TSAction and these modeling concepts cannot be considered separately. To show the duality of TSAction and TSState we have to extend the definition of TSAction from the previous subsection in order to show that TSActions changes the state of a system:

```
def TSAction{
 all a: TSAction                                  // for each TSAction a
          | some attr: Attrs                      // there is at least one attribute such that
          | some t1:a.instant_begin               // (if t1 = a.instant_begin) and
          | some t2: a.instant_end                // (if t2 = a.instant_begin) then
          | (t2 in t1.followingTE) &&             // [(t2 happens after t1) and
          (attr.attrState[t1] != attr.attrState[t2] )  // (attributes change their values in this TSAction)]
}
```

Code Fragment 10. TSAction (new definition)

Note that in this definition each TSAction changes the value of at least one attribute. To understand it, let's go back to the definition of state. To determine the sequence of TSActions in which an object can take part, TSState has to keep information about which TSActions are already executed, which TSActions are currently executed, and which TSActions can be executed in the future. Thus each TSAction changes at least one attribute in the TSState of an object. This attribute keeps information about the fact that this TSAction is finished (or not)[36].

Figure 66 shows the example of the model of state structure corresponding to the Alloy formal semantics.

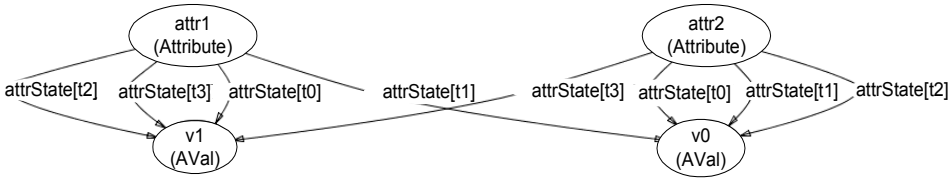


Figure 66. Example of the model of a system state, built with Alloy Constraint Analyzer

[36] This again shows the duality of state and behavior. Constraints of sequentiality are dual with the state information that tells which actions are finished (they specify the same thing from the point of view of behavior and state).

This example continues the example from Figure 65. It shows that a system has two attributes (attr1, attr2). Each attribute may have two values (v1, v0) in different time points (t0, t1, t2, t3). By analyzing two diagrams from Figure 65 and figure 66 we can see that the TSAction tsa2 changes the value of the attribute attr1 (*attr1.attrState[t1]=v0* and *attr2.attrState[t0]=v1*) and the TSAction tsa1 changes the value of the attribute attr2 (*attr2.attrState[t2]=v0* and *attr2.attrState[t3]=v1*).

2.3 Example of Complete Time Specific RM-ODP Model

The semantics of RM-ODP makes explicit how TSState and TSAction structures are related to each other. But the visual representation of models provided by Alloy Constraint Analyzer is not yet explicit enough. It represents TSState and TSAction structures separately (see Figure 65 and Figure 66). However, Figure 65 and Figure 66 are related by means of time points: any TSAction is defined between two time points (see code fragment 9) and any TSState is defined for a given time point (see code fragment 8). In order to explicit this relation between TSAction and TSState structures and to simplify Alloy diagrams, we use our notation. We use ovals to represent TSActions, rounded rectangles to represent TSStates. Each TSState is specified as a composition of TSStates of systems attributes. To represent time points we use small gray circles and to represent behavioral constraints we use stars. To represent relations between model elements we use arrows named in the same way as in Figures 65 and 66 with a slight difference. First, we do not show *corresponding_constraint* and *constrained_action* relations. We show only seq_constraint and next_actions relations that we use to indicate the sequence of actions. Second, instead of showing states of each attribute in a given time point, we show the state of all attributes together. For this purpose we use state existence relation. In our work we call diagrams built using this notation Time Specific RM-ODP diagrams. Figure 67 shows an example of such diagram that corresponds to the model automatically generated with Alloy Constraint Analyzer. This example is based on the models from Figures 65 and 66: the specification of the states of attributes from Figure 66 was added to the specification of behavior from Figure 65. The states of attr1 and attr2 are shown as parts of the composite states.

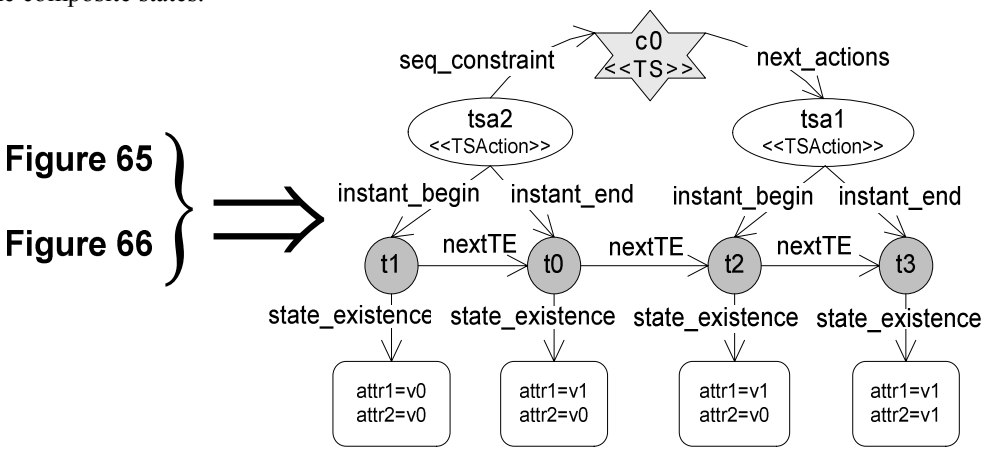


Figure 67. Time Specific RM-ODP model that combines the state structure from Figure 66 and the TSAction structure from Figure 65

We call a model specified with TSActions, a *Time Specific RM-ODP model*. As we can see a Time Specific RM-ODP model is precise but quite bulky (it contains too many details), even if the behavior to be modeled is simple. Fortunately, we can use a number of abstractions and simplifications to reduce the complexity of the model. Using simplifications can bring us to other different models. Further in our work we show some simplifications that can bring us to some existing modeling techniques: CCS process algebra, UML Statecharts and UML Activity diagrams.

3 Time Abstracted and Parametric RM-ODP Model

As we have seen in Section 2, Time Specific RM-ODP models have precise semantics that explain how different RM-ODP model elements are related to each other. However Time Specific RM-ODP models can not be used for modeling of the behavior with infinitely many TSActions. The behavior of an object may contain infinitely many TSActions due to the two following reasons. First, if the specification of the behavior is not limited in time. In this case, the sequence of actions would be unlimited. In order to make a finite specification of the infinite sequence of actions, we have to make an abstraction of time. In Section 3.1 we show how an abstraction of time can be done. Second, the specification of behavior may contain infinitely many actions if at some point in time only one TSAction is possible out of the infinitely many TSActions. For example, if an object receives

from its environment a single value out of infinitely many possible values, then using Time Specific RM-ODP model we have to specify a separate TSAction for each possible value. We have to do this because each TSAction can have only one post-state that would correspond to the reception of a concrete value. This will result in infinitely many TSActions and states of an object. In Section 3.2 we show how to deal with this problem by means of specifying parameterized actions.

3.1 Time Abstracted Actions

System designers often do not make explicit time information and keep only constraints of sequentiality. Sometimes the presence of time information makes modeling precise, however "the incorporation of concrete timing properties leads to a considerable loss of abstractness" [Broy91]. For example, using only TSAction does not allow specifying infinite behavior since it requires infinite sequence of TSActions. To make the specification of infinite behavior, we have to consider an abstraction of actual time information.

Based on the definition of TSAction, any TSAction changes the values of some attributes. We have also mentioned in Section 2 that any TSAction must change the value of at least one attribute. This attribute or attributes show the state of a TSAction (if this TSAction has been finished or not). We call attributes that show the state of a TSAction, *temporal attributes*. These attributes specify which TSActions can be executed next. Hence we call them temporal. All other attributes we will call *ordinary attributes*. In Alloy code we partition the set of all attributes to the set of temporal attributes and the set of ordinary attributes.

```
partition TAttrs, OAttrs :static Attrs          // attrubutes can be temporal or ordinary
```

For example, Figure 68 shows the example from the previous section where we distinguish between temporal and ordinary attributes.

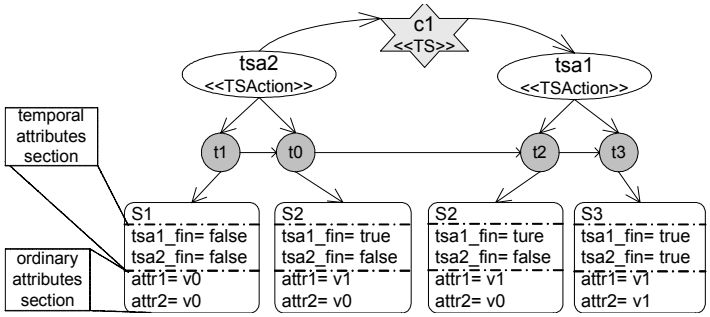


Figure 68. Time Specific RM-ODP model

Now we can define a predicate that characterizes the collection of TSActions that have the same result. For this purpose we use specification concepts presented in Section 2. Among specification concepts we use pre- and post-conditions[37]. In order to define a collection of TSActions with the same result, we will use TAPreconditions and TAPostconditions:

TAPrecondition: *precondition in the form: equals(attr,val) (or "attr = val"), where attr∈{ordinary attributes} and val∈{values of ordinary attributes}.*

TAPostcondition: *postcondition in the form: equals(attr,val) (or "attr = val"), where attr∈{ordinary attributes} and val∈{values of ordinary attributes}.*

```
pre_attributes: TAPrecondition -> OAttrs!   // one precondition specifies the value of one OAttrs
post_attributes: TAPostcondition -> OAttrs! // one postcondition specifies the value of one OAttrs
pre_values: TAPrecondition -> AVals!  // preconditions includes values for the ordinary attributes
post_values: TAPostcondition -> AVals!    // postconditions includes values for the ordinary attributes
```

Using pre and postconditions we can define a type of TSAction that we call: *Time Abstracted Action (TAAction).* We define it in the following way:

Type of TSAction (TAAction): *It is a type characterizing the set of TSActions: it specifies values of all ordinary attributes before and after any TSAction (that is an instance of TAAction). These values are specified with TAPreconditions and TAPostconditions.*

Instance of TAAction (): *TSAction that satisfies TAAction.*

Note that the definition of TAAction is a predicate stating that each TAAction requires ordinary attributes to have certain values before and after TSActions. Let's consider how this predicate can be expressed in Alloy.

---

[37] RM-ODP gives the following definition for preconditions and postconditions:
Precondition: *A predicate that a specification requires to be true for an action to occur.*
Postcondition**:** *A predicate that a specification requires to be true immediately after the occurrence of an action*

```
…TAAction, TAPreconditions, TAPostconditions…: SpecificationConcepts
satisfies_type(~type_for): TSAction -> TAAction+          // each TSAction has at least one type
TAA_preconditions: TAAction -> TAPreconditions+           // each TAAction has at least one TAPrecondition
TAA_postconditions: TAAction -> TAPostconditions+         // each TAAction has at least one TAPrecondition
```

Based on the definition of TAAction we have to require that each TAAction has TAPrecondition and TAPostcondition for each ordinary attributes (OAttrs):

```
def  TAAction{
  all taa:TAAction |
     all tsa:TAAction.type_for |                    // for all instancess of TAAction: tsa (TSAction)
     all attr:OAttrs |                              // and for any ordinary attribute attr
     one pre: taa.TAA_preconditions|                // there is one precondition for taa
     one post: taa.TAA_postconditions |             // there is one postconditions for taa,   such that
     attr = pre.pre_attribute  &&                   // (attr is an attribute of the pre precondition) &&
     attr = post.post_attribute &&                  // (attr is an attribute of the post postcondition) &&
    (one t:tsa.instant_begin |  attr.attrValue[t] = pre.pre_value) &&
                                // (the pre precondition specify the value of attr before TAAction) &&
    (one t:tsa.instant_end |  attr.attrValue[t] = post.post_value)
                               // (value of attrs is the same as the value of the precondtion)
 }
```

The class of TSAction is defined in the following way:

Class of TSActions*: A set of TSAction satisfying a TAAction type.*

To define formally the Class of TSActions, for each class we have to indicate which TSActions should be included in this class. In Alloy we can do this in the following way:

```
… TAAction_Class…: SpecificationConcepts          // TAAction_Class is a specification concept
associated_type: TAAction_Class!->TAAction!       // TAAction_Class has a corresponding type (TAAction)
member_of(~members): TSAction->TAAction_Class+    // each TSAction belongs to at least one TAAction_Class
     def TAAction_Class{
     all c:TAAction_Class |                       // for every TAAction_Class
c.associated_type in c.members.satisfies_type     // the type for the TAAction_Class is the same as the type for members of
                                                     this class
     }
```

To better illustrate these definitions we use the example from Section 2. In Section 2 the example was used to show a sample model generated by Alloy Constraint Analyzer. This model included two time specific actions (tsa1 and tsa2) and two attributes (attr1 and attr2). Here we show other elements of the model that we did not show in Section 2. These elements are specification concepts: TAActions, TAAction_Classes, TAPreconditions and TAPostconditions (See Figure 69).

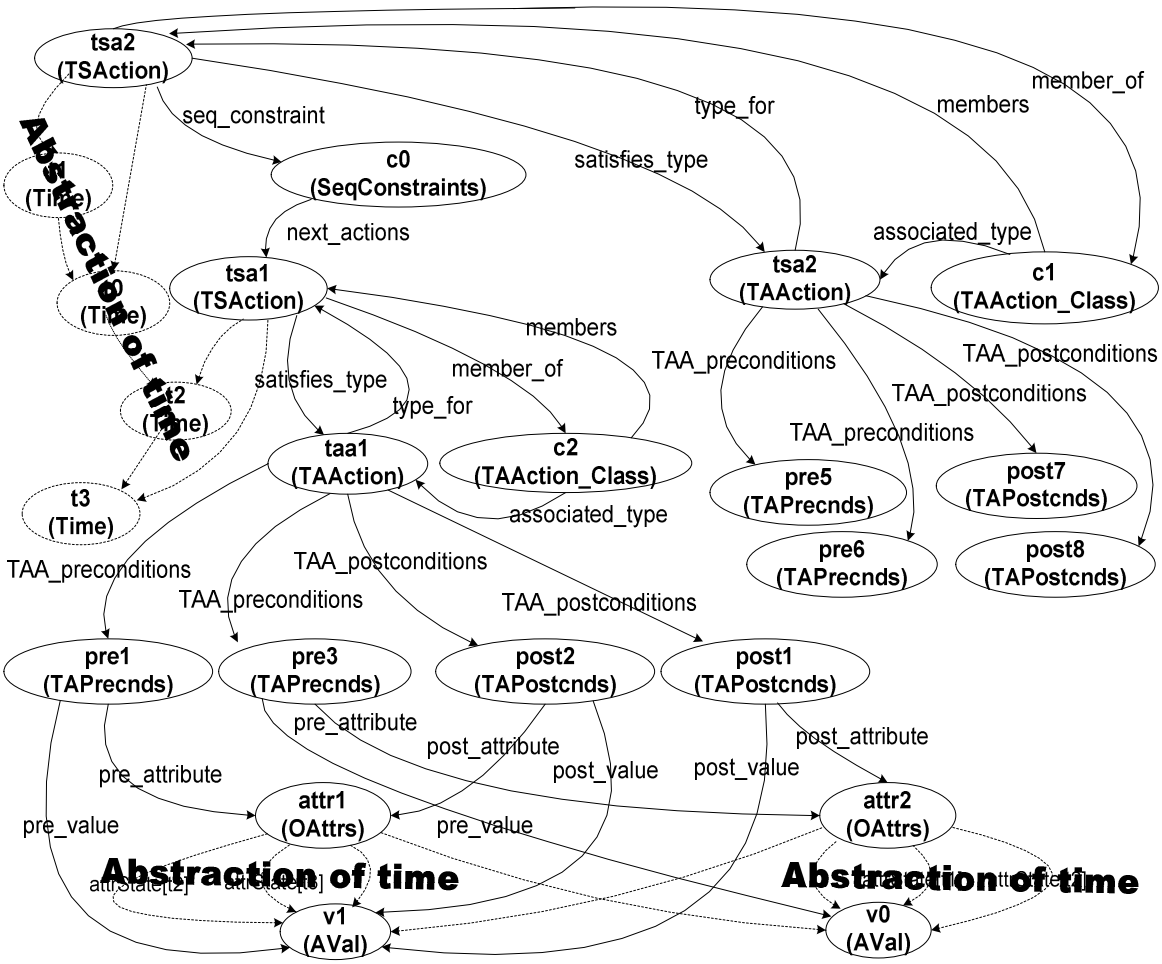
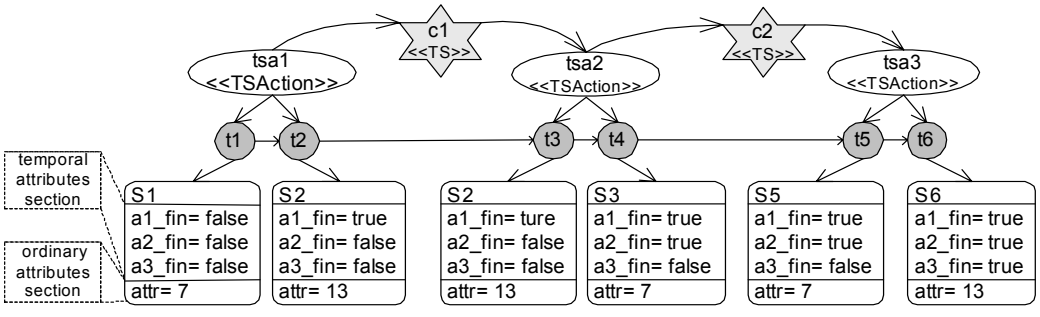


Figure 69. Example of the model of a system behavior and state built with Alloy Constraint Analyzer

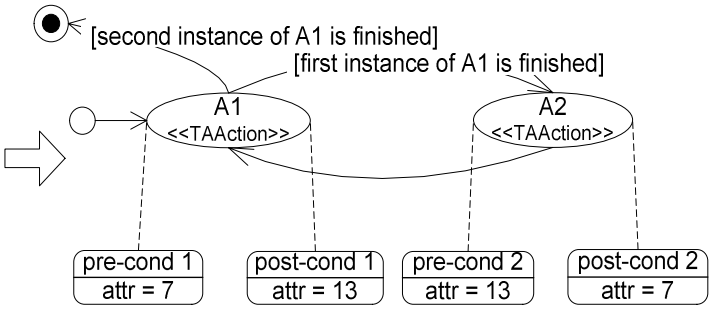
In the example from Figure 69, we suppose that the two attributes (attr1 and attr2) are ordinary attributes (we do not show temporal attributes in this example). To make reading of the model easier, we also do not show relations of TAPreconditions and TAPostconditions with attributes for the *tsa2* action.

The example form Figure 69 demonstrates how the abstraction of time can be done: instead of using TSActions (tsa1, tsa2) we can specify TAActions (that are types of TSActions). TAActions specify values for ordinary attributes before and after each TSAction. Thus the model of TAAction does not specify any particular time interval where it may occur and information about actual time intervals can be hidden (we show with dotted lines in Figure 69). This allows us to specify infinite behavior.

In order to do this we have to review the definition of TSBehavioralConstraints. We will use a concept of *time abstracted (TA) behavioral constraints*: constraints defined between TAActions. Therefore behavioral constraints of sequentiality define the sequence of TAActions such that this sequence preserves the sequence of TSActions: if two TSActions in the Time Specific RM-ODP model are sequentially constrained then two corresponding TAActions should also be sequentially constrained. Using TAActions and TA behavioral constraints brings us to the *Time Abstracted RM-ODP model*. To show how Time Abstracted RM-ODP model can be built based on the Time Specific model we use a slightly different example (see Figure 70) than the example from Figure 66.



(a)



(b)

Figure 70. RM-ODP diagram: From Time Specific RM-ODP model (a) to Time Abstracted RM-ODP model (b)

In the example from Figure 70.a we suppose that the TSActions tsa1 and tsa3 have the same TA preconditions (attr = 7) and TA postconditions (attr = 13). In that case they can be specified with TAAcion *A1* (see Figure 70.b). The TSAction tsa2 has to be specified with another TAAction *A2*.

Time Abstracted RM-ODP model does not include the partially ordered set of time points. It makes it possible to specify the infinite behavior of an object. But still we have to keep the information about the order of TSActions. To keep this information, we have to introduce two elements: initial and final points (black dot and black dot in a white circle). Another thing we should pay attention to is how to specify the constraints of sequentiality between TAActions. Any TAAction in Time Abstracted RM-ODP model may specify several TSActions, such that any TSAction has TSBehavioralConstraints with other TSActions. Thus we have to distinguish between instances of TAActions in order to specify constraints sequentiality correctly. The easiest way to do this is to introduce for each TAAction a counter that shows which instance of this TAAction has been finished. Based on this counter we can specify the sequence of TAActions. In case if some TAAction is followed by several TAActions, we specify conditions at corresponding arrows (for example "Second instance of A1 is finished"). Note that we simplified our notation for the constraints of sequentiality. We show them as arrows between sequentially constrained TAActions.

3.2 Parameterized TAActions

In the previous subsection we saw that by using TAActions we can specify a set of TSActions that assign the same values to ordinary attributes. But what about TSActions that assign different values to ordinary attributes but assign them in a similar way (based on some known mathematical function)? For them we can define TAPostcondition in the following way:

TAPostcondition (with parameter) [ver1]*: postcondition in the form: equals (attr,val) (or "attr=val"); where val: attr@pre→{values of ordinary attributes} and attr@pre is a value of attr before action.*

Here *val* is a unary function that takes the value of the attribute before action occurrence. TAPreconditions we can keep almost in the same form as before with the difference that *val* becomes a nullary function that can point to any element from the subset of ordinary attribute values:

TAPrecondition: *precondition in the form: equals (attr,val) (or "attr = val"), where attr∈{ordinary attributes} and val:__→{precondition values}⊂{values of ordinary attributes} is an unary function.*

In order to simplify our notation we will write these TAPreconditions in the form: "attr∈{precondition values}". Figure 71, for example, shows two TSActions (b1 and b2).

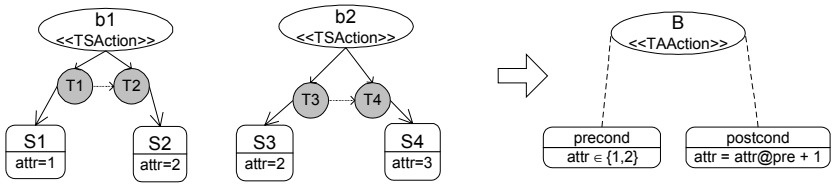


Figure 71. RM-ODP diagram: postcondition as a function

In this example we can define the TAAction B with TAPrecondition "attr ∈{1, 2}" and TAPostcondition "attr = attr@pre + 1". Thus we have defined TAAction with parameterized TAPostconditions. The parameter is the value of an ordinary attribute before the TSAction. In the similar way we can define TAPostcondition that takes TSAction as a parameter. This leads us to the concept of action with a parameter used in many modeling languages. Often a parameter is defined as value that can be passed to the object. For example, UML defines parameter in the following way:

Parameter [OMG01] *"is an unbound variable that can be changed, passed, or returned. Parameters are used in the specification of operations, messages and events, templates, etc. In the meta-model, a Parameter is a declaration of an argument to be passed to, or returned from, an Operation, a Signal, etc."*

Let's see what parameter means in RM-ODP terms. Figure 72 shows a set of TSActions from the Time Specific RM-ODP model $\{c_0, \ldots, c_N\}$. Only one of them can take place, depending on the choice of environment. Let's suppose that all these TSActions have similar TAPostconditions: these TAPostconditions differ only in the value that is assigned to the state attribute *attr*.

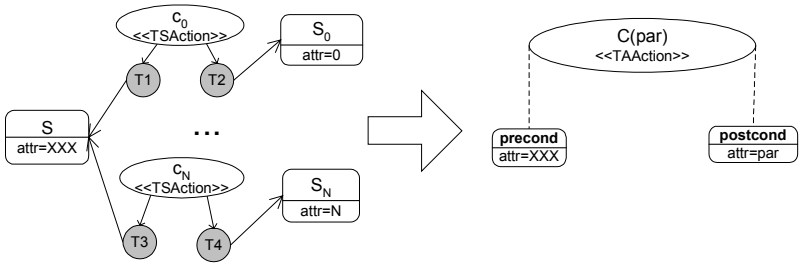


Figure 72. RM-ODP diagram: TAAction with parameters

To specify all these TSActions $\{c_0, \ldots, c_N\}$ with one type (one TAAction) we define TAPostcondition with a parameter [version 2]:

TAPostcondition (with parameters) [ver2]*: postcondition in the form: equals(attr, val) (or "attr = val"), where val: TSActionClass→{values of ordinary attributes} and TSActionClass⊂TSAction.*

In this definition *val* is a unary function that takes as an argument TSAction (from some TSActionClass) and returned a value to be assigned to the attribute *attr*. Based on the definition of TAPostconditions with parameters we can define TAAction (with parameter):

TAAction (with parameters): *It is a type characterizing the set of TSActions: it is a predicate that specifies values of ordinary attributed before and after any TSAction (that is an instance of TAAction). These valued are specified with TAPreconditions and TAPostconditions (with parameter).*

In the example, *C(par)*, par∈{0...N} in Figure 72 is TAAction that characterizes the set of TSActions $\{c_0, \ldots c_N\}$.You can see that we use a parameter *par* in the notation for TAAction. Thus *par* in Figure 72 allows us to relate a particular TSAction (the instance of TAAction) with a value assigned to the attribute *attr*.

In this section we considered different TAPreconditions, TAPostconditions and TAActions that have been defined using them. Many other TAActions can be defined by means of mixing the TAPreconditions and TAPostconditions presented in this section. For example, we can specify TAAction with mixed TAPostconditions: we can represent a value that is assigned to an attribute as an n-ary function: *val: $attr_1$@pre, $attr_2$@pre, TSActionClass →{values of ordinary attributes}*. Thus the value assigned to the attribute of an on object depends on: values of two attributes before TSAction and TSAction itself.

4 Mapping RM-ODP semantics with semantics of different specification languages

The abstraction of time considered in the previous section brings us to a Time Abstracted RM-ODP model. This model can be used as a generic model that we considered in the introduction. In this section we show how different views can be built on a genetic Time Abstracted RM-ODP model.

We begin with the example of the Time Specific RM-ODP model. We show how this example can be reduced to a Time Abstracted RM-ODP model using TAActions instead of TSActions. Then we consider how three views on the Time Abstracted RM-ODP model can be built. We show the three following view: CCS process algebra view, UML activity diagram view and UML statechart diagram view.

4.1 Example

Figure 73 shows the example of a Time Specific RM-ODP model. This model specifies the behavior of an object with nine TSActions. Five of them are TSInternalActions (they take place without the participation of the environment) and four of them are TSInterActions (they take place with the participation of the environment of the object). Names of TSInternalActions start with "a" and names of TSInterActions start with "e".

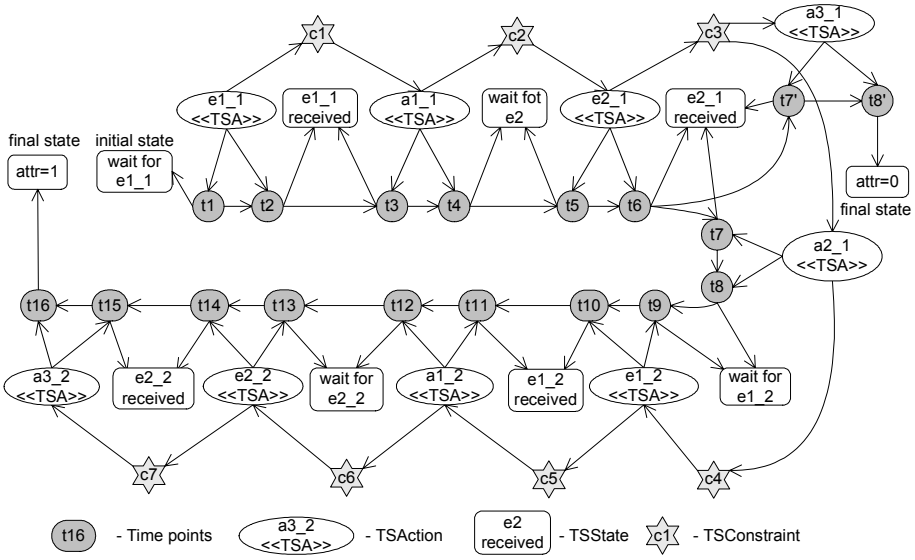


Figure 73. Time Specific RM-ODP model: an example of a behavior

The example shows the system TSStates before (pre-states) and after (post-states) each TSAction, TS Constraints and time points. You can see that a post-state after each TSAction is the same as a pre-state for the next TSAction. However, in general, these pre- and post-states can be different, since some other concurrent process can change the state of a system between two TSActions. Here we suppose that in our system there are no concurrent processes and thus there are no other processes that can change the state of the system between two consecutive TSActions. In this example we suppose that TSActions a1_1 and a1_2; e1_1 and e1_2; e2_1 and e2_2 have the same TA preconditions and TA postconditions. TSActions a3_1 and a3_2 also perform the same functionality (not specified here), with the slight difference that a3_1 makes the state attribute attr equal to 0, while a3_2 makes this attribute equal to 1.

First, if we make an abstraction of time. This brings us to the following Time Abstracted RM-ODP model:

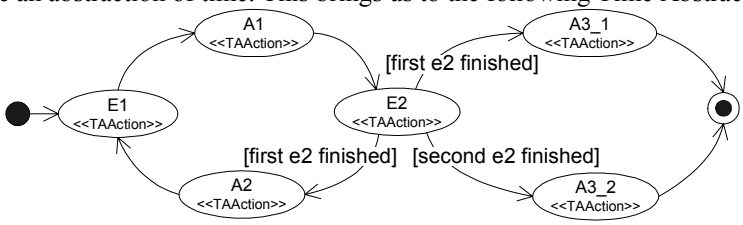
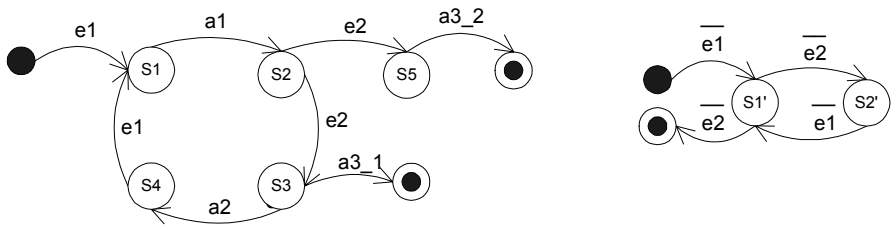


Figure 74. Time Abstracted RM-ODP model

Here A1, A2, E1, E2, A3_1 and A3_2 are *TSActions* that characterize the following collections of TSActions from Figure 73: {a1_1;a1_2}, {e1_1;e1_2}, {e2_1;e2_2}, {a3_1} and {a3_2} (for the purpose of simplicity we do not show TA preconditions and TA postconditions for these TAActions). Note that we introduced a counter for the action E2 and the conditions on the constraints of the sequentiality. It allows us to specify the same sequence of action instances in Figure 74 as the sequence of TSActions in Figure 73.

4.2 CCS Process algebra

In this section we consider how the Time Abstracted RM-ODP model can be transformed into a CCS [Milner99] model. First we explain how to build a CCS transition graph based on the RM-ODP model and then we show a corresponding CCS process expression. A transition graph can be built in the following way: any action becomes arc in the transition graph, constraints of sequentiality become states. Let's note that that just constraints of sequentiality become states in the transition graph but not pre- or post states. Some other concurrent process can change the state of a system between two actions. This means that the pre-state of an action and the post-state of a consecutive action can be different. But constraints of sequentiality in RM-ODP define exactly the same meaning as states in a transition graph: they specify the sequence of actions.



(a) **(b)**

Figure 75. Transition graphs of the system (a) and its environment (b)

Here also we have to pay attention to the transforming of constraints of non-determinism. In order to express them in the transition graph we have to model action e2 twice. It shows that the system makes the internal choice between the two "branches" of behavior without being influenced by its environment. For a better illustration we also show the behavior model for the system environment. An interaction of an object with its environment can be represented as a reaction between the following pairs of actions and their complements {e1,$\overline{e1}$} {e2,$\overline{e2}$}. The same specification in the form of concurrent process expressions would be:

$System = e1.S1$

$S1 = a1.S2$ $\qquad Environment = \overline{e1}.S1'$

$S2 = e2.S5 + e2.S3$ $\qquad S1' = \overline{e2}.S2' + \overline{e2}$

$S3 = a2.S4 + a3\_1$ $\qquad S2' = \overline{e2}$

$S4 = e1.S1$

$S5 = a3\_2$

Note, that the transition graph in Figure 75 does not allow us to count instances of action e2. Thus e2 in Figure 75 can have more than two instances. To have only two instances (e2_1 and e2_2) we have to specify them separately without grouping them into one action.

4.3 RM-ODP and UML Statechart and Activity Diagram

The further simplification (using modeling of actions with parameters) of our example from Figure 74 leads us to the behavior model shown in Figure 76, where the post-condition for action a3(p) is "attr = p". We use the model in Figure 76 to show how UML Activity and Statechart views can be defined.

Note, that in all behavior models we considered above, interactions and internal actions are modeled using the same notation (the sign of oval). But UML uses a slightly different notation. UML has the two following terms:

> **(UML) Action:** "An action is a specification of an executable statement that forms an abstraction of a computational procedure that results in a change in the state of the model, and can be realized by sending a message to an object or modifying a link or a value of an attribute" [OMG01].
>
> **(UML) Event**: "An event is a noteworthy occurrence. For practical purposes in state diagrams, it is an occurrence that may trigger a state transition" [OMG01].

Although there is no direct mapping of an RM-ODP interaction and an RM-ODP internal action with a UML Action and UML Event, in our particular example we can conclude that E1 and E2 correspond to UML events and A1, A2, A3 correspond to UML actions. "An event is something done to the object; an action is something that the object does" [Stevens00]. An event in UML is considered as an action trigger and modeled in the way it is shown in Figures 77 and 78.

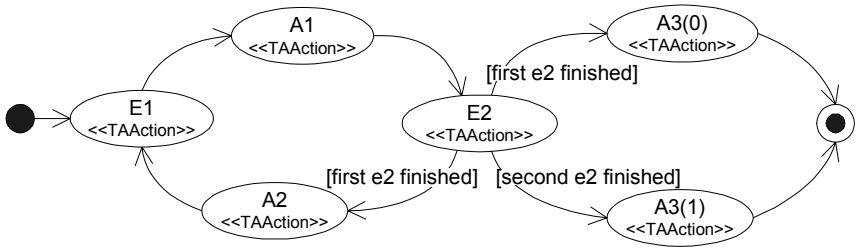


Figure 76. RM-ODP diagram: Simplification of the model using actions with parameters

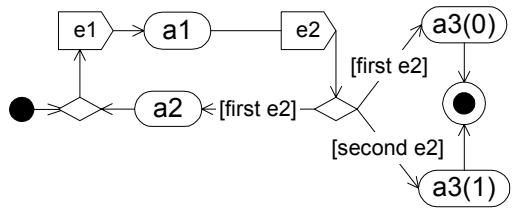


Figure 77. UML activity diagram

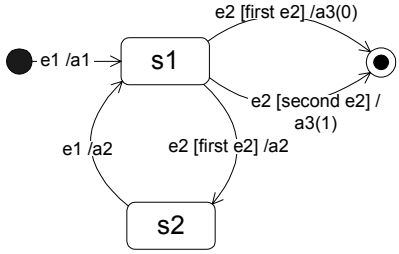


Figure 78. UML statechart diagram

5 Conclusion

In this appendix we have analyzed the possibility of using RM-ODP Part 2 "Foundations of the Open Distributed Processing" as a generic semantic domain for systems behavior modeling. We have considered the minimum set of RM-ODP concepts that a system designer needs for "any kind of modeling activity" [ISO96]. These concepts form the generic semantic domain for system behavior modeling and allow a system designer to specify generic behavior models.

RM-ODP behavior models are based on the concept of Time Specific Action (TSAction) and Time Specific State (TSState). Time Specific Actions directly represent things that happen in the Universe of Discourse with explicit reference to time. An object in each time point is specified with one Time Specific State. We call a model that use TSActions and TSStates, a Time Specific RM-ODP model. However, "the incorporation of concrete timing properties leads to a considerable loss of abstractness" [Broy91]. To make Time Specific RM-ODP models more abstract and to be able to specify the infinite behavior, we considered a Time Abstracted RM-ODP model. A Time Abstracted RM-ODP model makes an abstraction of time by means of using Time Abstracted Actions (TAActions) and Parameterized TAActions. TAAction characterizes the set of TSActions that assign the same value to some ordinary attributes of an object. Parameterized TAAction characterizes the set of TSActions whose postconditions can be specified as a mathematical function.

We believe that a Time Abstracted RM-ODP model can be used as a generic behavior model. Having a generic behavior model allows a system designer to define different views on this model, where each view addresses particular problems that a system designer wants to solve. Each view may have its specification language. In this work we considered how a Time Abstracted RM-ODP generic model can be seen from the three views done with the following specification languages: CCS process algebra, UML Activity Diagram and UML Statechart Diagram. We explained the mapping of corresponding concepts from the semantic domains of these three languages and from the generic semantic domain based on RM-ODP.

This work continues the work done by Naumenko [Naumenko01] that formalizes the semantics of RM-ODP. The main contribution of this work is the formal definition of TAAction. We show a formal relation of Time Specific Behavior with Time Abstracted Behavior. This relation can be used in case tools to check the consistency between behavior instance diagrams (like UML Sequence Diagram) and behavior type diagrams (like UML Activity diagrams). The definition of Time Abstracted Action is based on the definition of State Structure and Composite State. These concepts extend the notion of composition presented in RM-ODP. RM-ODP defines the composition of object and the composition of behavior. However RM-ODP does not define how the state of the composite object can be defined. In this work we define the Composite State that can be used to specify a state of the composite object.

Laboratory of Systemic Modeling (LAMS)
EPFL, LAMS-IC, INN - Ecublens, CH-1015 Lausanne, SWITZERLAND
http://lamspeople.epfl.ch/balabko/
Tel. +41 21 693 6793; Fax. +41 21 693 6610
Pavel.Balabko@epfl.ch

Balabko Pavel

32 years old, born 02.12.1972, married, nationality: Russian

Key Expertise System analysis and design, conceptual modeling, business process modeling, software engineering, Enterprise Architecture (EA), IT architecture.

Professional Skills

System Analysis and Design	**RM-ODP** (ISO/ITU standard for system modeling), **UML** (Unified Modeling Language) • 5-years of experience
Enterprise Architecture	**TOGAF**(The Open Group Architecture Framework), **Zachman** framework, **RAD** (Role Activity Diagrams), **IDEF0**, **IDEF1**, **IDEF6** (NIST standards for functional, information and design rational modeling) • 2-years of experience **SEAM** (Systemic Enterprise Architecture Method) • 4-years experience **Situation Based Modeling Framework in the context of EA** • Author of the framework
Software Development	**RUP** (Rational Unified Process), **KobrA** (component-based application development), **Java**, **J2EE, Rational Rose, OORAM** (Object-Oriented Role Analysis and Modeling), **Telelogic TAU UML Suite**, **Together**, **AOP** (aspect-oriented programming)**.** • 1-3-years of experience
Management	**FlowTeam-Method** (an approach based on principles of self-organization for unleashing the collective intelligence of a work team) • 2-years experience
System Support	**Certified Novell Instructor**, since 1996 **Certified Novell Engineer**, since 1997

Professional Experience

2001 - 2005	**PhD Researcher (LAMS-IC-EPFL, Switzerland)** PhD thesis: developed the Role Based Modeling Framework in the context of EA. Major in: EA, concern-based system analysis and design, Business Process Modeling Framework features: ▪ Separation of concerns ▪ Explicit design rationale in the development of enterprise systems ▪ Represents enterprise systems as the hierarchy of levels ▪ Integrated with (AOP) Aspect Oriented Programming Publications: 8 papers in refereed conferences and books; The PhD thesis defense is panned for the end of March 2005
1999 – 2002	**Research and Teaching Assistant (ICA-EPFL, Switzerland)** Teaching Assistance: ▪ Assisted in the development and teaching of the "Object Oriented Analysis and Design" undergraduate course (1-semester course, more than 100 students; given in winter semesters of 1999-2002) ▪ Supervised 6 Postgraduate/Semester/Diploma projects (including industrial projects in collaboration with Novartis and SkyGuide);

	IT architecture design: ▪ Performed the analysis of the on-line stream retailer business model. ▪ Designed the generic UML-based IT architecture of the on-line Stream Retailer system for the "StreamCom" project (6 research groups, more than 10 project members, 2-years)
1997 - 1998	**Project Engineer ("Eurotour" in collaboration with Industry & Construction Bank, St-Petersburg, Russia)** ▪ Designed the prototype of the groupware/workflow system for the Industry & Construction Bank (5 project members, 2-years)
1995 – 1998	**Novell Trainer and Education Manager (State Education Center, St-Petersburg, Russia)** ▪ Organized and gave more than 100 Novell courses and seminars; ▪ Published multiple technical reviews and marketing papers in Saint-Petersburg IT journals ▪ Provided Technical Consulting to customers of the education center
1995-1996	**Member of "Graphicon 95, 96" organization committee (State Education Center, St-Petersburg, Russia)**

Education

2005	PhD thesis defense is planned for the end of March 2005
1999	Doctoral school certificate, EPFL, Lausanne, Switzerland
1996	M.S. in computer science, State Saint-Petersburg Technical University, Russia
1990	School-leaving certificate, Saint-Petersburg Physical and Mathematical Lyceum

Languages

English: fluent
French: fluent
Russian: native speaker

Personal Interests

Hiking (participated in trips to Ural, Caucasian, Carpathian mountains), dancing (participated in demonstrations together with K'DANSE, Switzerland), alpine skiing

Scholarships and Grants

- Doctoral school scholarship, May 1998 (competition rate: 1/14)
- Grant for students from Saint-Petersburg Government, December 1995