SeamCAD:
a Hierarchy-Oriented Modeling Language and a Computer-Aided Tool for Entreprise Architecture

THÈSE N^{O} 4225 (2008)

PRÉSENTÉE LE 12 NOVEMBRE 2008
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE MODÉLISATION SYSTÉMIQUE
SECTION DES SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Lam Son LÊ

acceptée sur proposition du jury:

Prof. C. Petitpierre, président du jury
Prof. A. Wegmann , directeur de thèse
Prof. C. Atkinson, rapporteur
Prof. S. Spaccapietra, rapporteur
B. Wood, rapporteur

Lausanne, EPFL
2008

Abstract

Modeling Enterprise Architecture requires representing multiple diagrams of an enterprise, which typically shows the multiples business entities, IT systems, even software components and the services they offer. This could be done by a team of stakeholders having different backgrounds. One way to do this is to structure the model into hierarchical levels each of which can be of interest of just some, not all, stakeholders. Due to the differences in their background, stakeholders - the modelers may not want to use a single modeling approach, even a widely-recognized one, to build the enterprise model, which can be shared by the whole team. Developing a modeling framework that can be applied uniformly throughout the entire enterprise model and that can be used by all stakeholders is challenging. First, the framework should have a uniform approach to specifying the services offered by business entities, IT systems and software components and to describing their implementation across hierarchical levels. Second, the framework should allow the stakeholders to represent the service specification and the service implementation of multiple business entities and IT systems, even within the same hierarchical level. Third, the services offered by those entities and systems should be represented at different levels of granularity. Last but not least, the modeling framework should maintain the well-formedness of the enterprise model and the consistency between different diagrams opened by different stakeholders of the team.

Today, there exist a few modeling methods or development processes in the field of Enterprise Architecture, as well as in software and system modeling that can address these issues to some extent. Among them, Adora, KobrA and OPM best meet the aforementioned four criteria, although they were not initially developed for modeling Enterprise Architecture. As a study on the state of the art, we analyzed these methods with respect to the four aforementioned modeling challenges. In this thesis, we define a modeling language and present a computer-aided tool for modeling Enterprise Architecture hierarchically. This modeling language allows the modeler to structure an enterprise into hierarchical levels, in terms of both organization and services. The computer-aided modeling tool helps the modeler visually build her model across levels and brings all levels together to make a coherent, well-formed model. Enterprise models can be visually built and represented in a notation that is based on the Unified Modeling Language using this tool. The modeling language is formally defined in Alloy – a lightweight declarative language based on first order logic and set theory. The data manipulated in the tool is verified against the Alloy code that formalizes the language. The modeling language and the computer-aided modeling tool constitute a hierarchy-oriented framework called SeamCAD that specifically address the four aforementioned issues. This framework has been applied several projects, both in industry and academic settings. We evaluated it by inviting external practitioners, researchers and master's students in our university to use it and to give their feedback.

Keywords

Enterprise Architecture, System Modeling, Computer-Aided Enterprise Modeling, Reference-Model of Open Distributed Processing, Unified Modeling Language, Alloy

Résumé

Modéliser une architecture d'entreprise exige la représentation de multiples vues de l'entreprise qui, typiquement, montrent les entités métiers, les systèmes informatiques, même les composants logiciels et les services offerts. Le modèle est construit par une équipe d'intervenants provenant de domaines différents. Une façon d'y parvenir est de structurer le modèle de manière hiérarchique. A chaque niveau correspond des intervenants différents. L'élaboration d'un environnement de modélisation qui peut être appliquées de façon uniforme dans tout le modèle d'entreprise et être utilisé par toutes les parties prenantes est un défi. Tout d'abord, l'environnement devrait avoir une approche uniforme pour représenter les entités métiers, les systèmes informatiques, et les composants logiciels. Deuxièmement, l'environnement devrait permettre de modéliser plusieurs entités métiers et systèmes informatiques, à la fois comme boîte noire ou boîte blanche. Troisièmement, le comportement des entreprises, des systèmes et même des composants logiciels et de l'interaction entre ceux-ci devraient pouvoir être représentés à différents niveaux de granularité. Enfin, l'environnement de modélisation devrait permettre de maintenir la cohérence du modèle d'entreprise et la cohérence entre les différentes vues ouvertes par les membres de l'équipe.

Aujourd'hui, il existe quelques méthodes de modélisation ou processus de développement qui peuvent répondre en partie à ces questions. Parmi eux, Adora, KobrA et OPM sont les mieux aptes à répondre aux quatre critères ci-dessus. Aucun d'entre eux n'a été développé pour l'architecture d'entreprise. Le travail présenté dans cette thèse définit un langage de modélisation hiérarchique pour l'architecture d'entreprise. Ce langage de modélisation permet de structurer la représentation de l'entreprise et de son environnement dans les niveaux hiérarchiques organisationnels et comportementaux. La gestion du modèle se fait au moyen d'un outil informatique de modélisation qui permet de construire son modèle à différents niveaux (organisationnels ou fonctionnels). La notation graphique s'inspire de UML. Le langage de modélisation est défini dans une façon formelle en Alloy - un langage déclaratif basé sur le logique de premier ordre et la théorie des ensembles. Le langage de modélisation et l'outil informatique constituent l'environnement appelé SeamCAD. Il résout les quatre questions mentionnées ci-dessus. Cet environnement a été appliqué dans une étude de cas lié à un cours donné par notre groupe et dans plusieurs autres projets. Il a également été évalué en invitant des praticiens externes et des étudiants de niveau master de notre université à l'utiliser et à donner leurs commentaires.

Mots clés

Architecture d'entreprise, Modélisation de Système, Reference-Model of Open Distributed Processing, UML, Alloy

TABLE OF CONTENTS

Preface

Graphical modeling is about creating a visual representation, which can help people understand the domain they want to model better than they do with a textual description. Today, problems in different domains can be described in model-based representation. Drawing diagrams and maintaining them are essentially important in modeling. People who doing modeling may consider them modeling burdens due to large number of elements and diagrams they are dealing with. However, these burdens can be relieved if an appropriate computer-aided modeling tool is put in use. This is similar to the case of programming code. Programmers usually need an integrated development environment with which they can correctly write code and efficiently maintain it.

By the time that the research work presented in this dissertation commenced, I participated in a small project to find a specific computer-aided tool that could meet the modeling criteria set by xC – the obsolete name of the modeling method developed by professor Alain Wegmann – my advisor and his research group. There were two main criteria. First, it should be possible to work with multiple diagrams; some of which should be object diagrams (as opposed to class diagrams). Second, the tool should allow the user to draw a diagram in which she can represent people, systems and collaboration between them altogether. We made a survey of more than 20 modeling tools to see how they could be used for xC and unexpectedly came to the conclusion that no tool, at that time, met our requirements. Note that UML 2 did not exist yet at that time. Some of the UML-based modeling tools support the object diagram, but all of them do not allow putting an actor pictogram, a class (or a package or a subsystem) together with a collaboration pictogram in a diagram. In a use-case diagram, we can have both an actor and a use-case. In a class diagram, we can have both a class and a pattern structure. The user-case and the pattern structure can perfectly represent collaboration. However, putting the three together in a diagram was simply impossible. In addition, we noticed that, although it was possible to work with multiple diagrams on the same model, the consistency between these diagrams was questionable. Apparently, the tools manage a list of weakly-related diagrams (each of which has a set of model elements) instead of maintaining a coherent model containing related model elements. For example, a UML actor can appear in a use-case diagram, a class diagram or a sequence diagram. However the user of the tool has almost no means to state that they are simply multiple appearances of the same model element – a person who interacts with the system.

We decided to develop a specific computer-aided tool for xC and gradually changed the name of the method to SEAM. The goal was to be able to model IT systems, business entities, people, the collaboration between them as well as their internal design. We began to call this tool SeamCAD – accordingly to the name of SEAM and the term Computer Aided Design though later we learnt that it was not really a CAD tool. We defined the modeling terms for the SEAM method based on the Reference-Model of Open Distributed Processing. The tool SeamCAD should support the modeling terms of the SEAM method.

As suggested by my advisor, I made several prototypes of the tool by drawing all possible diagrams for a specific case-study and linked them together through HTML pages. I was amazed by the number of diagrams created for a quite simple case-study. Finding a systematic way to link them and to efficiently navigate among them was challenging. Eventually we found out that we needed to define rules to do this. I started to realize that developing a computer-aided tool and defining a modeling language was inseparable. This is how the topic of this doctoral thesis was born.

In the years that followed, we implemented the SeamCAD tool and the SeamCAD modeling language in a mutual way. As soon as we improved to the tool, we discovered some feature that was not defined yet in the language. We then added them to the definition of the language. This was in turn beneficial to the implementation of the tool in the sense that rigorous definition of the language can be translated into an appropriate design of the data structure in the tool. For instance, to represent the collaboration between IT systems and business entities at different levels of granularity, we need to elaborate their representation to show several sub collaborations. We then faced a problem: which model elements that emerge in the detailed representation of the systems should be represented together with certain collaboration? We found the following solution: the notion of granularity level was applied not only to the collaboration but also to the model elements that emerge in the representation of the IT system or the business entity. As such, binding relations are established between the model elements that emerge in the IT system or the business entity and a collaboration that is at the same level of granularity. These binding relations were mapped to the references between Java objects representing model elements in the implementation of the tool. An algorithm was developed to generate diagrams that correctly render model elements at a given granularity level. On the other hand, improving the tool to implement newly-defined features in the language is a good way, if not the best, to prove that the language practically works and I felt rewarding a lot in doing it.

Defining a modeling language requires doing meta-modeling. It usually ends up with a meta-model consisting of a diagram that expresses building blocks of the language and a list of well-formedness rules. While trying to come up with such a meta-model for SeamCAD, I gradually took the following principle although I could not prove it formally: the simpler the diagram is, the more complicated the well-formedness rules are. In other words, if the building blocks are loosely expressed, more well-formedness rules and more statements in each of these rules are needed. I found it interesting to strike for the balance between the tightness of the specification of the building blocks and the amount of well-formedness rules stated.

It is worth mentioning the relationship between the tool implementation and the language specification. The tool was fully implemented in Java (using only standard Java libraries). The modeling language (more precisely, the meta-model of this language) was formally specified in Alloy – a formal declarative specification language developed by Daniel Jackson and his group at MIT based on first order logic and set theory. I enjoyed formalizing the meta-model of SeamCAD in Alloy in the sense that this language still has an object-oriented syntax although it is basically a declarative language. However, due the unlikeness in the nature of the two languages (one is imperative, the other is declarative) and in the richness of the user-interface they support (one implements a graphical editor, the other relies on a tool do visualization), the Alloy code metrics and the Java code metrics are significantly different. I was surprised by the fact that the Alloy code is nearly 300 times more compact than the Java code in terms of the number of lines of code, and 25 times in terms of the number of classes. What I learnt from these differences was that automating (even partially) the implementation of the tool based on declarative specifications is still a very difficult (and interesting too) problem. In fact, what I successfully did on this matter is doing data verification on the instance models that are manipulated in the tool against the Alloy specification code that formalizes the modeling language.

Acknowledgements

I wish to express my deepest appreciation and gratitude to my all people that contributed to the success of this Ph.D. work. I am most grateful to Professor Alain Wegmann – my advisor, for sponsoring and inspiring me continuously, for giving me complete freedom in time management, and for continuously challenging me with fabulous questions. I could thus perform my research from its beginning to its completeness in an efficient, yet enjoyable way. For me, the experiences of tackling research problems by trying alternative approaches, of reporting research results without any personal opinion, of intensely debating on different modeling and programming paradigms and of positioning research topics in a broader context were absolutely unique.

I profoundly acknowledge the members of my Ph.D. jury (listed in an alphabetical order): Professor Colin Atkinson - the leader of the Software Engineering Group at the University of Mannheim for challenging me in positioning my doctoral work in the field of SOA and modeling business processes, Professor Stefano Spaccapietra – the head of the Database Laboratory in EPFL for commenting on the way the SeamCAD modeling language was defined in the pre-defense and on the terms that are used for formulating the motivation problems, Bryan Wood – a very active researcher on RM-ODP from the United Kingdom for thoroughly reading this dissertation and pointing out the diagram-to-code inconsistencies and the insufficient explanations in the dissertation that was submitted before the defense. I am thankful to Professor Claude Petitpierre (EPFL) for chairing my thesis committee.

I would like to express my gratitude to the practitioners, the researchers in the field of Enterprise Architecture and some of my colleagues who helped me validate my research results by practicing the SeamCAD tool and giving their opinions frankly. They are Mr. Henry Peyret – a senior analyst from Forrester Reasearch, Dr. Thomas Baar from Tech@Spree, Mr. Frédéric Bouchet from Rolex, Mr. Patrick Fleury – the director of DOP Gestions SA, Dr. Andrey Naumenko from Triune Continuum Enterprise, Mr. Alexander Samarin – a consultant in Enterprise Architecture, Dr. Kerstin Langenberg, Dr. Jiyong Zhang, Mr. Jan Schönbächler. I am also thankful to (listed in an alphabetical order) Megha Agarwal, Florence Le Goff, Yi Liu, Nebil Mansour, Hugo Marcelo Muriel, Erik Ragnerstam, Luis Téran, Murielle Ange Tiambo and Anmol Tomar. They are master's students at EPFL who worked with me to validate the SeamCAD tool in a cooperative, yet friendly manner.

I am deeply thankful to Mrs. Danielle Alvarez, Mrs. Angela Devenoge, Mrs. Patricia Hjelt and Mrs. Evelyn Duperrex for their great administrative support, Mr. Jean-Pierre Dupertuis and his colleagues for their countless professional support on the computer systems I used for more than six years at EPFL and Mrs. Holly Cogliati for the wonderful corrections on the English in my research papers.

It would be incomplete if I did not mention the people that helped me realize the opportunity of doing doctoral work at the School of Computer and Communication Sciences, EPFL - one of the greatest university institutes in Europe. I would like to thank Dr. Nguyen Thanh Son from the University of Technology of Ho Chi Minh City and Mr. Daniel Gorostidi – director of ELCA, for encouraging me to pursue this doctoral work. I am also thankful to Mrs. Annette Jaccard for her counsel on social matters she kindly gave to the foreign students of EPFL during the first two years of my new life at Lausanne, Switzerland.

I wish to share a big hug with my EPFL colleagues and my friends in Switzerland – some still work at EPFL and others have left it, as well as those I knew before heading for EPFL and still keep in touch with. I will definitely miss the fruitful discussions that I had with Pavel Balabko (on RM-ODP terms and the course OOAD), José Diego de la Cruz (on declarative programming in Alloy), Andrey Naumenko (on ODP-based modeling), Gil Regev (on XML-based marshalling and client-server architecture), Irina Rychokva (on meta-model definition) and Amir Talaei-Khoei (on validating my research results with undergraduate students). We also had many interesting chats on different topics that were not necessarily related to research nor teaching. In the life outside of EPFL, I very much enjoyed the way my friends shared in the spare time amazing hobby interests on various subjects including football, hiking, computer games, photograph, windsurfing, scenic driving, sightseeing, travelling, skiing… Making a full list of them here is not possible. I would like to thank them all.

From the bottom of my heart, I am deeply grateful to my parents and my wife for so many priceless things and so much great love they have always given to me. My mother and my father have always given their spiritual support to any achievements not only during the period I pursued my highest education in Switzerland, but also during my student age in Vietnam. The more I mature, the more I realize that I have benefited a lot from their teaching and pedagogical professions in my childhood. My wife, Thi Tran Chau Nguyen, has accompanied me in the later stage of my Ph.D. I am thankful to her for whatever she did behind the scenes that silently contributed to the completion of this long-term work.

Chapter 1: Introduction

***Overview**: Enterprise Architecture captures the whole vision of an enterprise in the various aspects regarding both business and IT resources. We consider Enterprise Architecture as a discipline that analyzes the services offered by an enterprise and its partners to the customer, the services offered by the enterprise to its partners and the organization of the enterprise itself. During an Enterprise Architecture project, a team – typically a multi-disciplinary team - develops an enterprise model that represents the enterprise, its environment and its internals. It is possible to represent the enterprise model hierarchically. However, it is challenging to do so. Most notably, the team should: (1) have a uniform approach to modeling the specification and the implementation of services provided, across organizational levels, by all business entities and IT systems of the enterprise; (2) represent the service specification and the service implementation of multiple business entities and IT systems in the enterprise; (3) model the services of the business entities and the IT systems at different functional levels; (4) achieve the well-formedness of the enterprise model and the consistency between its views. It is our goal in this dissertation to develop a modeling language and a computer-aided tool for modeling Enterprise Architecture, which specifically address the aforementioned challenges.*

1.1. Enterprise Architecture and Modeling

Enterprise Architecture (EA) captures the whole vision of an enterprise in various aspects regarding both business and information technology (IT) resources [1]. In EA, the goal is to align the business resources and IT resources to maintain or improve the competitiveness of the enterprise. EA is a discipline that analyzes the services offered by an enterprise and its partners to the customer, the services offered by the enterprise to its partners and the organization of the enterprise itself and of its IT. Making an EA project can, for example, help the enterprise gain more customers, reduce the operation costs or increase its agility. This can be done by better identifying the services that the enterprise provides to the customer, by removing the duplication and inconsistencies in business processes and/or information flow, by giving the management more IT-supported facts for making decision with ease.

During an EA project, an EA team – typically a multi-disciplinary team - develops an enterprise model that represents the enterprise, its environment and its internals. The representation of the enterprise can include various aspects such as the services offered by the enterprise, by the IT systems, as well as their implementation in terms of business processes and IT application. Working with a model is important. When making the model, the team develops an agreed and shared representation of the enterprise, of its environment and of its internals. They also define what the project needs to achieve.

There are different approaches of modeling EA. The enterprise model can represent business and IT resources in different ways, for example, as a network, a hierarchy or even in an ad-hoc manner. Because people tend to reason in terms of hierarchy [2], representing the enterprise as a hierarchy is a convenient way to structure and to build the enterprise model. In our research group, we focus on the hierarchical method. In such an approach, model elements can be organized into levels of containment. For example, we can model the way an enterprise provides its customers with services by representing the enterprise as a value network that consists of several companies that collaborate to fulfill the services offered by the value network. We can drill down through the service hierarchy by representing each company as a set of departments collaborating with one another; each company has IT applications and people; etc.

1.2. An Example

To illustrate typical difficulties the EA team face when developing an enterprise model, we give a hypothetical example in this section. The example describes a bookstore whose management decides to provide the company's services via the Internet. The management has a goal to specify the services that the bookstore can provide its customers with and to describe how to implement them using business and IT resources. The management creates an EA team in charge of this project. Figure 1 presents a simplified representation of the organization and services of the enterprise using ad-hoc notation. In this figure, a regular rectangle represents a business entity or an IT system or a software component. A rounded rectangle can be attached to a regular rectangle to represent the main service offered by the business entity or the IT system drawn under the regular rectangle. The smile symbol stands for people. The lines connecting these entities and people denote the containment hierarchy. The entire representation given in Figure 1 can be interpreted as follows. A book-selling market contains a `Bookstore Value Network` and a `Customer`. The bookstore value network consists of there companies: a bookstore company named BookCo (responsible for the service of processing the orders placed by the customer), a shipping company called ShipCo (responsible for shipping the books ordered) and a publishing company PubCo (responsible for supplying the books that were ordered but not yet available in the inventory of the bookstore company). The departmental structure of the bookstore company shows two departments: one for coping with the purchasing data (PurchasingDep) and the other for managing an inventory (WarehouseDep). The purchasing department has IT support (OpApp) and some Clerk who operates it. To fully analyze the impact of this project, the EA team has to reason about the multiple levels shown in Figure 1 (i.e. the market level, the value network level, the company level, the department level, and the IT level) in order to analyze the services offered by the business entities and the IT system (e.g. the service of Selling book offered by the Bookstore Value Network to the Customer, Buying book offered by the Customer to the Bookstore Value Network, Shipping book offered by the ShipCo to the BookCo and PubCo) and to build a structural representation of the whole enterprise that would commonly be shared by the members of the EA team.

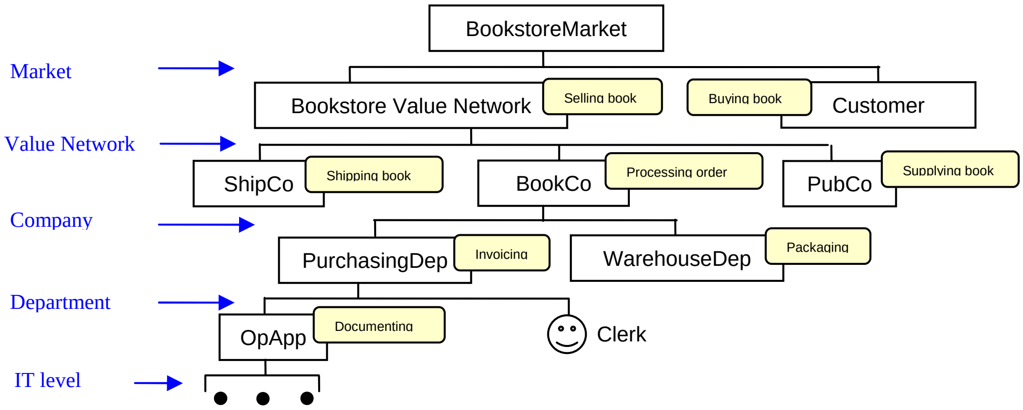


Figure 1. Initial representation of the enterprise model of the online bookstore

In this project, the EA team essentially models the business entities, the IT system (drawn under regular rectangles in Figure 1) and their environment, the services provided to the customer by these entities, the company to company (and department to department) business processes, information flow and interaction between the IT system and the clerk and possibly the overall architecture of the IT system. The following challenges are noticeable

- How can the services Selling book, Buying book, Processing order, Shipping book, Supplying book, Invoicing, Packaging and Documenting, which are offered by Bookstore Value Network, BookCo, ShipCo, PubCo, PurchasingDep and OpApp, be specified within these business entities in this model? The business entities and the IT system mentioned belong in fact to different levels that are indicated in Figure 1. Can the same modeling technique be applied for market, value network, company, IT level…?
- Given that the aforementioned services can be specified in this enterprise model, can BookCo, ShipCo, PubCo and the collaboration between them be represented to describe the implementation of the service Selling book, which is offered by Bookstore Value Network? Similarly, how can PurchasingDep, WarehouseDep and the collaboration between them be represented to describe

the implementation of the service `Processing order` that is offered by `BookCo`?

- How can these services be represented at different levels of granularity? For instance, the service `Selling book` provided by the `Bookstore Value Network` can be represented as whole or as a composition of three constituent services that specifically deal with getting order, payment and delivery.
- Members of the EA team may need different diagrams that render different parts of the enterprise model of the online bookstore. For instance, `BookCo` may appear in a diagram made by a member who is interested in defining the company to company business processes. It may also appear in another diagram opened by another member whose interest is to define the internal structure or the behavior of `BookCo`. How can we ensure that the two diagrams are always consistent in rendering `BookCo`?

It is important for the EA team to solve these challenges in order to make an enterprise model for the online bookstore. These concrete challenges will be generalized as motivation problems in the next section.

1.3. Motivation Problems

Defining a common modeling framework that can be applied uniformly across hierarchical levels is challenging. The following four modeling challenges are generalized from the concrete ones identified in the previous subsection

- Uniformness: To have a uniform approach for modeling the specification and the implementation of services provided by all business entities and IT systems across hierarchical levels showing the organization of the enterprise
- Multi-entity: To represent the service specification and the service implementation of multiple business entities and IT systems in the enterprise
- Granularity: To model services of the business entities and the IT systems at different levels of granularity
- Well-formnedness: To maintain the well-formedness of the enterprise model and the consistency between its diagrams

These four challenges will be referred to throughout this dissertation. The state of the art in EA will be analyzed using criteria that are formulated from them. Our contribution, in this thesis, is solutions to these problems.

1.4. Framework for Modeling Enterprise Architecture Hierarchically

Today, there exist a few modeling methods or development processes in the field of Enterprise Architecture, as well as in software or system modeling, which can address these issues to some extent. Among those that meet the aforementioned four criteria to some extent. KobrA, OPM and Adora are the best although they were not initially developed for modeling Enterprise Architecture. A solution that specifically addresses the aforementioned issues is needed. In our research group, we define a method for modeling EA called SEAM [3] [4] [5]. This method can be used for doing requirement engineering of an enterprise. Within SEAM, a modeling framework that consists of a hierarchy-oriented modeling language and a computer-aided tool was developed. An overview of this framework is presented in this subsection.

Contribution: a Hierarchy-Oriented Modeling Language for EA

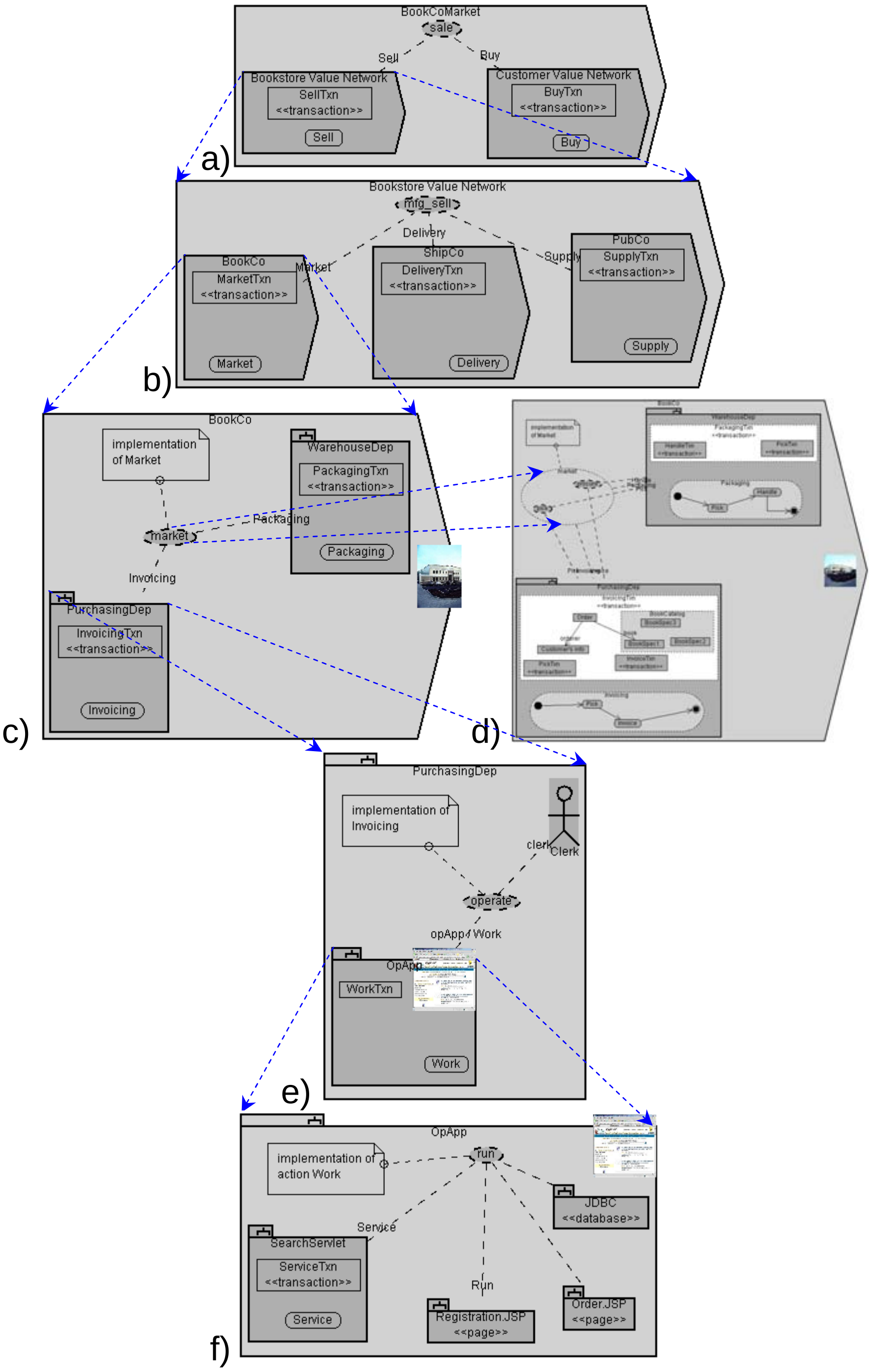


Figure 2. Representation of the bookstore example in the modeling language defined

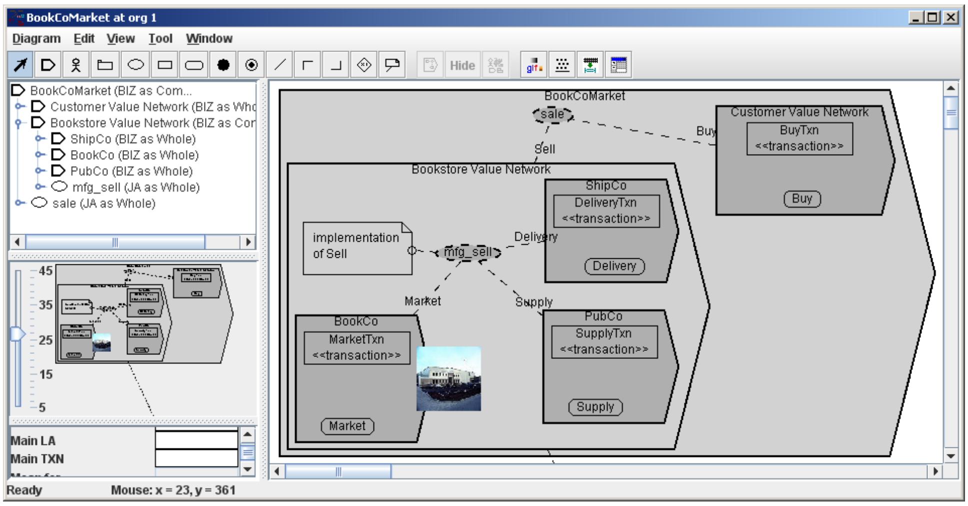
Using the hierarchy-oriented modeling language that will be defined in this dissertation, a model for the online bookstore can be built as illustrated in Figure 2. This modeling language provides the modeler with building blocks to represent all business entities and IT systems, as well as software components in the enterprise model of the online bookstore. The business entities such as the book-selling market, the customer, the value network of the bookstore are diagrammatically represented under block arrows. The IT systems and software components, such as the web application in the purchasing department of the bookstore company, are drawn under the pictogram of the UML (Unified Modeling Language) subsystem. The collaboration between these entities and systems are drawn under dashed ellipses. The properties and the behavior of each business entity or IT system can be represented under the UML class and action pictogram. In terms of hierarchical level, Figure 2 a) represents the market level; Figure 2 b) the value network level; Figure 2 c) and 2 d) the company level; Figure 2 e) the department level and Figure 2 f) the IT level. Note that the difference between Figure 2 c) and Figure 2 d) is the level of granularity of how the services are specified (the challenge *granularity* is addressed).

Note that Figure 2 has dashed arrows that connect one pictogram from one diagram to another pictogram in another diagram. The two pictograms, which are connected by a pair of dashed arrows, represent the same business entity or IT system (or some collaboration between them) but at different hierarchical levels. Overall, these arrows illustrate the hierarchy of the enterprise model of the online bookstore. Throughout the hierarchical levels that are illustrated from Figures 2 a) to 2 f), each business entity or IT system can be specified either as whole or as composite. A pair of dashed arrows runs from the pictogram that represents a business entity or an IT system as whole to it seen as composite. An entity or a system seen as whole shows externally-observable properties and actions that together characterize the services it offers to other entities or systems. An entity or a system seen as composite consists of other entities and systems that together show how the offered services are implemented. This illustrates how the challenges C1 and C2 are addressed.

Contribution: a Computer-Aided Modeling Tool for Modeling EA

All the diagrams given in Figure 2 are taken from the computer-aided tool that was developed as part of the modeling framework. The tool allows the modeler to visually build the entire enterprise model of the online bookstore. Figure 3 is a typical screenshot of this tool, which shows two things: an overview of the enterprise model in the top-left corner and a diagram representing some part of the model. The overview is shown in a tree-view widget and the diagram is shown in a graphical panel. The diagram represents the market level and the value network level. This widget and panel are interactive in the sense that the modeler can browse and edit the enterprise model of the bookstore by interacting with them. For instance, if the modeler opens the enterprise model of the online bookstore in the tool, she can get any diagram of Figure 2 by using the browsing functionality of the tool. The modeler can also edit the diagrammatically represented part of the enterprise model of the bookstore in this window.

In the tool, data structure of the enterprise model are engineered to capture not only model elements that can be seen in the diagrams of Figure 2 but also the references between them (for example, the link between the bookstore value network and the bookstore company, the publishing company and the shipping company). The tool does not manage a list of diagrams. Instead, it generates diagrams from this data structure according to the modeler’s needs. The consistency between diagrams can thus be achieved. This explains how the challenge *well-formedness* is addressed.



Ready   Mouse: x = 23, y = 361

Figure 3. The enterprise model of the online bookstore shown at the market level and the value network level in a window of the computer-aided tool

1.5. Research Methodology

The modeling framework presented in this dissertation consists of two main parts: the definition of a modeling language and the development of a computer-aided tool. Figure 4 positions this framework as a research work in the Hevner's Design Science Research in Information System [6]. The knowledge base used for this research work includes the Living Systems Theory [7], the Reference Model of Open Distributed Processing [8], the SEAM method [3] [4] [5] and the state of the art on EA and related fields. Two computer languages are used as the methodology for this research: Alloy as formal declarative language (used for the formalization of the modeling language) and Java as imperative programming language (used for the development of the computer-aided tool).

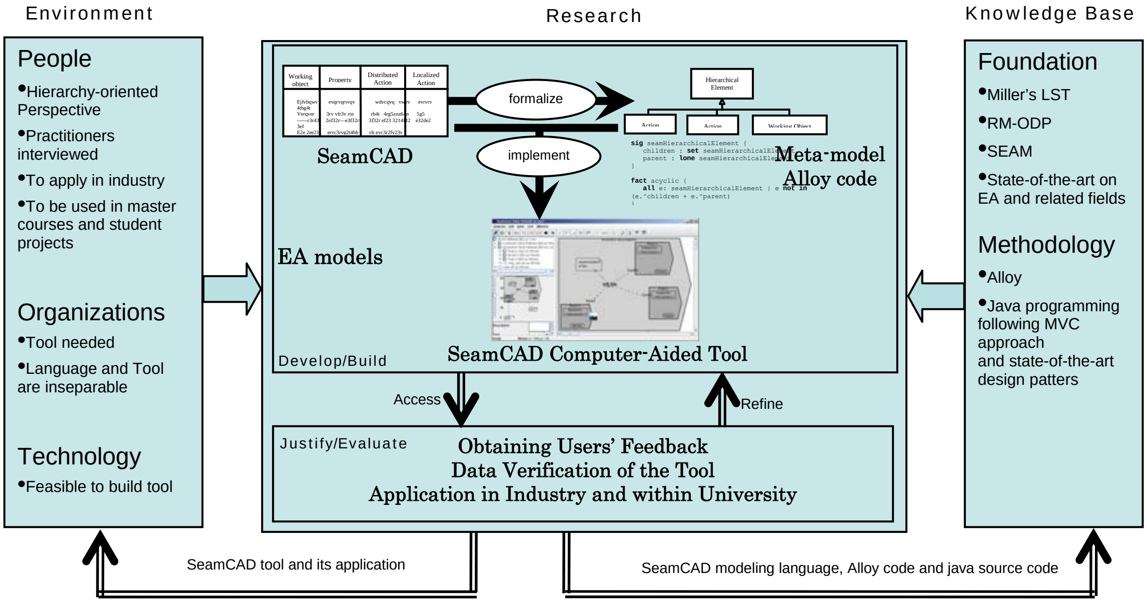

Environment
Research
Knowledge Base
People
•Hierarchy-oriented Perspective
•Practitioners interviewed
•To apply in industry
•To be used in master courses and student projects
Organizations
•Tool needed
•Language and Tool are inseparable
Technology
•Feasible to build tool
Working object
Property
Distributed Action
Localized Action
formalize
implement
SeamCAD
Hierarchical Element
Action
Action
Working Object
Meta-model
Alloy code
EA models
Develop/Build
SeamCAD Computer-Aided Tool
Access
Refine
Justify/Evaluate
Obtaining Users' Feedback
Data Verification of the Tool
Application in Industry and within University
Foundation
•Miller's LST
•RM-ODP
•SEAM
•State-of-the-art on EA and related fields
Methodology
•Alloy
•Java programming following MVC approach and state-of-the-art design patterns
SeamCAD tool and its application
SeamCAD modeling language, Alloy code and java source code

Figure 4. The proposed framework for modeling EA hierarchically

In the research block (see Figure 4), we define a modeling language (both informally and formally) and we develop a computer-aided tool. Together, they are called SeamCAD. The informal definition verbally explains the building blocks of the SeamCAD modeling language and how they can be used for building an enterprise model. The building blocks were initiated in the SEAM method that was developed in our research group. The formal definition of SeamCAD includes a meta-model (including a UML class diagram and a list of well-formedness rules) and its formalization in Alloy - a formal declarative modeling language based on the first-order logic and the set theory [9]. The SeamCAD computer-aided tool basically implements the SeamCAD modeling language. This tool allows an EA team to build their common enterprise model using the building blocks following the well-formedness rules of the SeamCAD modeling language. It also enables the members of the EA team to generate diagrams to view their enterprise model at different hierarchical levels. The data structure that represents enterprise model in the SeamCAD tool was verified against the Alloy code that formalizes the SeamCAD modeling language.

To evaluate our research work, we invited practitioners, researchers and master's students to evaluate the modeling language and the computer-aided tool. They used SeamCAD and then give their evaluation and suggestions in a questionnaire. In addition, SeamCAD was applied in several projects for building enterprise models and a case-study in a master's course given by our group.

The definition and development of SeamCAD augments our knowledge base. Concretely, the Alloy code that formalizes the SeamCAD modeling language and the Java open source code that implements the SeamCAD tool serve as a basis for further researches in our group. Its applications in the industry sector and academic settings are impacts of this work to the research environment block (see Figure 4).

1.6. Research Scope

As the dissertation is positioned in the filed of EA that can, in principle, include various modeling aspects. In this section, we clarify the scope of this dissertation by mentioning which aspects it addresses and which ones it does not.

The main objective in developing SeamCAD was to help the EA team in documenting and discussing their enterprise model from a functional standpoint. The members of the team can build a common enterprise model in SeamCAD, which can serve as the centralized documentation of their project. Thanks to the SeamCAD tool, the members of the EA team can navigate in the enterprise model and open diagrams that show some part of the common enterprise model so that they can discuss their project or make a project report out these diagrams. However, SeamCAD does not support model execution. Although the order between services can be specified in SeamCAD, the semantics of services is not defined formally enough to create an executable model. It is considered as future work of this thesis. Appendix A gives an example of how the semantics of actions can be specifed in SeamCAD.

This dissertation is about the SeamCAD framework that consists of a modeling language and a computer-aided tool. However, it does not describe a modeling method. In fact, SeamCAD can be regarded as one of the modeling languages and computer-aided tools for SEAM – an EA modeling method developed at our group.

It is necessary to mention that the SeamCAD modeling language and tool do not address the non-functional requirements. Specifically, the following aspects are not addressed by the enterprise models created in SeamCAD: finance, governance, business rules, quality of services, security, database, network, system interoperability and low-level software design. The enterprise model created in SeamCAD is mainly about the services and the organization of the enterprise being modeled.

1.7. Outlines

This dissertation is structured as follows: Chapter 2 analyzes the state of the art in modeling methods that are related to Enterprise Architecture; Chapter 3 presents the SeamCAD modeling language; Chapter 4 describes the computer-aided tool that was specifically built for the SeamCAD modeling language; Chapter 5 discusses certain applications and evaluations of the contributions presented in Chapter 3 and Chapter 4. Chapter 6 draws some conclusions and points out future research directions. Appendix A presents the formalization in Alloy for the online bookstore model that is diagrammatically represented throughout the chapters. The materials used for obtaining users' feedback are described in Appendix B. Appendix C analyzes different computer-aided modeling tools that can be used for modeling EA hierarchically.

In Chapter 2, different modeling methods in the domain of enterprise architecture and software/system development are analyzed, with respect to criteria formulated based on modeling challenges that were identified for modeling EA hierarchically. Among the analyzed methods, the three that best meet these criteria will be compared to the SeamCAD modeling language in a more detailed grid that consists of not only hierarchy-related issues but also fundamental modeling aspects such as black-box and white-box.

Chapter 3 is dedicated to the SeamCAD modeling language. First, the foundation of the modeling language and its relation to the RM-ODP are introduced. Then the building blocks of the modeling language are informally defined in English. This informal definition is followed by a meta-model of SeamCAD that consists of a diagrammatic descriprion of building blocks and a list of well-formedness rules. Next, the meta-model of the SeamCAD modeling language is formalized in Alloy - a lightweight, declarative language based on set theory and first-order logic. The last section of Chapter 2 presents the notation used in the SeamCAD modeling language and tool.

In Chapter 4, the computer-aided tool that was specifically developed for the SeamCAD modeling language is presented. This chapter begins by discussing the role of such a tool in modeling hierarchical systems in EA and concludes with a list of requirements a computer-aided tool for modeling hierarchical systems should fulfill. Subsequent sections in this chapter explain the tool SeamCAD and show how this tool can meet the identified requirements. Next, the way diagrams of SeamCAD are generated from a coherent model and their automatic layout are described. The last two sections of Chapter 4 address two proof-related issues: how to trace the design and the implementation of the SeamCAD tool back to the meta-model of the SeamCAD modeling language and how to verify that the models manipulated in the SeamCAD tool match the Alloy code that formalizes the SeamCAD modeling language.

Chapter 5 presents some applications of SeamCAD and how it was validated by practitioners and students. In the research group where this Ph.D. work was carried out, SeamCAD was applied in several projects some of which were in conjunction with industry. An enterprise model was built using SeamCAD for the case-study of a master's course given by our group. This case-study is about a company who manufactures and sells lightweight aircraft engines. There is also an application made with a company in which an enterprise model is built to manage sale processes and customer relations of the company in the market of watch-parts manufacturing. Another project used SeamCAD in making an enterprise model of one of the department building on our university campus. This model was useful for specifying how the building should be equipped and what IT system should be installed in the new building. An additional project was set up to investigate the possibility of furthering the model created in SeamCAD to be able to simulate System Dynamics. Evaluations and feedback for SeamCAD from 20 participants, including practitioners, researchers in EA and our master's students are also presented in Chapter 5.

This dissertation has three appendixes. Appendix A formalizes a diagrammatic model built using the SeamCAD modeling language and its computer-aided tool in Alloy. This appendix exemplifies the ideas for future researches that are pointed out in the conclusion remarks. Appendix B presents the materials that were used for obtainning evaluations and feedback on SeamCAD from practitioners, researchers and students. These materials include a tutorial that helps practitioners in EA-related domain use the tool and some slides that were used for working with students to validate the SeamCAD tool and two questionnaires that were used for obtaining feedback from practitioners and students, respectively. Appendix C analyzes existing tools in the field of modeling enterprise, software and system.

Chapter 2: State of the Art

***Overview**: Today there exist many modeling methods and frameworks. They can be classified into three categories: enterprise modeling, system modeling and software modeling. Knowing to which extent they can represent Enterprise Architecture hierarchically is important for developing a hierarchy-oriented modeling framework for Enterprise Architecture. For this purpose, we evaluated them in terms of four criteria that are necessary in order to represent Enterprise Architecture hierarchically. Among the analyzed methods, the three methods that best meet these criteria will be compared in a more detailed grid that consists of not only hierarchy-related issues but also fundamental modeling aspects like black-box / white-box specification and state modeling.*

Today there exist many modeling methods and frameworks. They can be categorized into enterprise modeling, system modeling and software modeling. In this chapter, they are evaluated in terms of

- expressing organizational hierarchy
- multi-system representation
- expressing behavioral hierarchy
- well-formedness of model and consistency between diagrams

The aforementioned criteria are originated from the four modeling challenges: uniformness, multi-entity, granularity and well-formedness that are presented in Chapter 1, Section 1.3

2.1. Software and System Modeling

Many methods have been developed for system engineering and software engineering. We describe here some of the approaches that can be used for modeling EA hierarchically. We selected the methods that have features closest to the aforementioned criteria. They are (listed in alphabetical order) Addora, Catalysis, KobrA, OPM, SysML, UML Profile for EDOC and UML 2. Note that UML is not a method but a modeling language.

Adora [10] is an object-oriented modeling method that features hierarchical decomposition and the integration of all aspects in one coherent model. Objects in Adora are composite by default. The organizational hierarchy can be reasoned in terms of the composition of objects. This hierarchy is visually depicted in the tool of Adora. The behavioral hierarchy can be thought of as tree-like hierarchy of scenarios (in Adora, a scenario is similar to a use-case in UML). The hierarchy of scenario is not visible in the tool, however. Adora is a system-centric modeling language.

Catalysis [11] is a component-based development process that analyzes and designs in three levels: business, IT system and software components. It uses its own notation that is inspired from UML. Catalysis put a lot of effort in making behavior refinement. It made popular the notion of two kinds of action, namely joint action and localized action. In principle, the organizational hierarchy of Catalysis is visible in the containment hierarchy, which is typically up to three levels: context level, software level and component level. As Catalysis defines development process for software-intensive systems, it is a typical system-centric method. In Catalysis, the well-formedness of model is maintained by keeping traceability between different refinements.

KobrA [12] proposes a recursive model that describes IT systems/components. KobrA takes the notation from UML. In KobrA, each component is described dually in terms of specification and realization. The recursive approach of KobrA implicitly suggests that this method can deal with as many organizational levels as the modeler wishes to. But in practice, KobrA aims at representing only the context level and some component levels. As KobrA tackles software development process, it is a typical system-centric method although the concept of component can be used for representing any business entity or IT system in an enterprise model. In KobrA, it is up to the modeler to maintain the well-formedness of her model by practicing inter-diagram and intra-diagram rules.

Object-Process Methodology (OPM) addresses the modeling of systems in general [13]. It has its own notation and provides a modeling tool called OpCat [14]. The building blocks of OPM are object, process, state and relations. The organizational hierarchy can be described via object aggregation and the behavioral hierarchy can be reasoned in terms of process aggregation. Multiple systems can be designed in the same OPM model. Although OpCat supports hierarchical modeling by allowing the modeler to zooming in a specific object or process, it is up to the modeler to maintain the well-formedness of her model by making sure that all diagrams are consistent.

Systems Modeling Language (SysML)[1] is developed by the OMG. It is based on UML. SysML targets the design of large industrial systems (e.g. aircraft, power plants). In SysML, there are two organizational levels: the context of the system to be developed and the internal structure of the system. Like most of UML-based modeling language, SysML is system-centric. The behavior of the system to be developed can be refined using UML state-chart and activity diagrams. The well-formedness of a SysML model is up to the extent to which diagrams are kept in synch by the modeler.

UML[2] is widely-used modeling language for representing software-intensive systems. As such, UML is a system-centric modeling language and addresses organizational levels not higher the context of software system, which is normally shown using use-case diagrams. Although it is possible to represent multiple software components or subsystems in a UML model, most of the project is built around one system that is typically called the system of interest. Behavioral hierarchy can be reasoned by in terms of actions and states in activity diagrams and state-chart diagrams. The well-formedness of a UML model depends on the extent to which diagrams are kept in synch by the modeler. The UML profile for Enterprise Distributed Object Computing[3] takes a step further by defining organizational structure of components, making the organizational hierarchy visible.

Table 1 summarizes the evaluation of software and system modeling methods that are listed in an alphabetical order.

Table 1. Evaluations of system/software modeling methods

Method	Organizational Hierarchy	Multiple Systems	Behavioral Hierarchy	Well-formedness of Model
Adora	✓	x	statechart & scenarios	✓
Catalysis	Software level Component level	x	✓	✓
KobrA	✓	✓	✓	x
OPM	✓	✓	✓	x
SysML	System level Component level	x	✓	x
UML Profile for EDOC	✓	✓	✓	x
UML 2	System level	one system, many components	✓	x

The following limitations can be seen from the software/system development methods that were analyzed

- The number of organizational levels is generally limited. However, Adora, KobrA and OPM can have as many levels as the modeler wants.

[1] OMG System Modeling Language, http://www.sysml.org/

[2] Unified Modeling Language, http://www.uml.org/

[3] UML profile for EDOC, http://www.omg.org/technology/documents/formal/edoc.htm/

- Most of methods are system-centric meaning they focus on one system, typically an IT system.
- UML-based methods have weakly-related diagrams. The modelers are responsible for the well-formedness of their model.

2.2. Enterprise Modeling

A number of methods have been developed for modeling enterprises. We analyze the methods have features closest to the aforementioned criteria C1), C2), C3) and C4). They are (listed in alphabetical order) Archimate, BPMN, CIMOSA, DEMO, IDEF, TOGAF and Zachman.

Archimate proposes an integrated modeling framework for Enterprise Architecture including organizational structure, business processes, information systems and infrastructure [15]. This framework proposes 3 layers, namely business layer, application layer and technology layer. Each of these layers can further be divided into sub layers, making the organizational hierarchy virtually visible. The process can be broken down into smaller processes, implicitly showing the behavioral hierarchy. Archimate claims that different aspects of enterprise architecture are integrated in a single model, but it does not discuss clearly how to maintain the well-formedness of model.

Business Process Modeling Notation (BPMN)[4] provides business users with a rich notation for modeling business processes. The processes defined in BPMN are hierarchical. Nevertheless, BPMN doesn’t address the organizational hierarchy although multiples systems can be shown without any hierarchy in different pools of a BPMN diagram. It is straightforward to maintain the well-formedness of a BPMN model if the processes are organized into a tree-like structure.

The Computer Integrated Manufacturing Open System Architecture (CIMOSA, also known as the ISO EN/IS 19440 standard) focuses on the modeling of processes in the context of computer integrated manufacturing projects. CIMOSA defines four modeling views: function view, information view, resource view and the organization view. The resource view and the organization view address the structure of resources (humans, machines, information systems…) but do not show an explicit organizational hierarchy. Multiples systems can be represented in CIMOSA. The model well-formedness is not specifically addressed.

Design & Engineering Methodology for Organizations (DEMO) is a method for (re)designing organizations [16]. DEMO defines three types of models of an organization: the black-box model, the white-box model, and the flow model. The black-box model deals mainly with the external behavior of a system and supports the functional refinement. In the flow model, a system is conceived as a network of nodes transforming the input flows into output flows. The white-box model defines the constructional refinement of the system. Multiple systems can be represented. The organizational hierarchy and the model well-formedness are not discussed however.

IDEF[5] (Integrated DEFinition Methods) is a set of methods that address many aspects of enterprise modeling (e.g. function, data, process, object-oriented design). It could be considered as a method for organizations to analyze and clearly state their

[4] OMG Business Process Modeling Notation, http://www.bpmn.org/

[5] Integrated Definition Methods, http://www.idef.com/

information resource management needs and requirements. Multiple systems can be elaborated in IDEF but no concept equivalent to the organization level is proposed. As IDEF focuses on data modeling rather than behavior modeling, the behavioral hierarchy is not addressed neither.

TOGAF[6] and Zachman [17] propose ad-hoc modeling frameworks in which multiple systems can be represented. But they do not have any explicit organizational or behavioral hierarchy. The well-formedness of model is down to the burden of the modeler in making her model.

Table 2 summarizes the evaluation of enterprise modeling methods that are listed in an alphabetical order.

Table 2. Evaluations of enterprise modeling methods

Method	Organizational Hierarchy	Multiple Systems	Behavioral Hierarchy	Well-formedness of Model
Archimate	Layers	✓	process decomposition	integrated model
BPMN	Pool & Lane	✓	✓	✓
CIMOSA	Cell & Unit & Element	x	✓	x
DEMO	x	✓	✓	x
IDEF	x	✓	x	x
TOGAF, Zachman	Ad-hoc	✓	Ad-hoc	x

The following limitations can be seen from the enterprise modeling methods that were analyzed

- The number of organizational levels is generally limited.
- Behavior modeling is missing in some methods such as IDEF
- The modelers are responsible for the well-formedness of the model they created

2.3. A Comprehensive Comparison of Adora, KobrA and OPM

In this section, the three methods that best meet the evaluation criteria presented in Section 2.1 and 2.2 are analyzed in details though the example of the online bookstore. They will be compared to our modeling language in a more comprehensive grid than the one used in Section 2.1 and 2.2.

2.3.1. Adora

Adora is a requirement engineering modeling method that features hierarchical decomposition and the integration of all aspects in one coherent model. The modeling language of this method has the following building blocks: object, state, scenario, role and link. It comes with a tool (a prototype) that helps building Adora model. Figure 5 gives the grammar of the Adora modeling language.

```
Specification
specification ::= { specification fragment }
specification fragment ::= { model element }
model element ::= object | state | scenario | relationship | annotation
Objects/States/Scenario declarations
object ::= object i | singleton object e | object set e
```

[6] The Open Group Architecture Framework http://www.togaf.org/

```
system object ::= object i | state | scenario | singleton object e | object
set e
object i ::= singleton object | object set
singleton object ::= object name [obj body] end
singleton object e ::= object name “: external” end
object set ::= object set name [obj body] end
object set e ::= object set name “: external” end
env object ::= element of the environment name
obj body ::=
{annotation} [attributes {attribute}1]
[operations {operation}1] [contains {system object}1] attribute ::= . . .
operation ::= . . .
state ::= pure state | start state
pure state ::= state name [contains {state}1] end
start state ::= start state end
scenario ::= scenario name [scenario body] end
scenario body ::= . ..
```

Figure 5. An excerpt of the EBNF formalization of Adora [18]

Figure 6 illustrates the model of the bookstore created in the Adora tool. To the left is a diagram that shows 3 organizational levels. To the right is an overview of the model. `BookCoMarket` object is at the top organizational level; `Bookstore Value Network` and `Customer` second organizational level and at the third level there are `BookCo`, `ShipCo`, `PubCo` object. Note that Adora takes the box-in-box notation for objects. The interaction between objects is represented via roles. `Bookstore Value Network` takes the `Seller` role and `Customer` takes the `Buyer` role. These two roles participate in the scenario `sale`, which is extended to three other scenarios: `procure`, `pay` and `deliver`. The internal structure of an object and its state can dually be represented in the same diagram. For example, inside the pictogram of `Bookstore Value Network`, `BookCo`, `ShipCo`, `PubCo` object are visible. It three main states `order received`, `money received`, `shipping` are also visible inside the pictogram. Also visible is a scenario `mfg_sale`. The `shipping` state can be broken down further into `book loaded` and `book delivered`. In terms of component objects, `BookCo` has `WarehouseDep`, `PurchasingDep`, `Inventory` and `Books` as component objects. Note that only `Books` is object set, the others are just singleton object. This difference is well reflected in the pictograms used in the diagram. The Adora tool manages a coherent model and show it in a tree-like view that we can wee in the right of the Figure 6. The diagram can easily be customized by interacting with pictograms. For instance, the `BookCo` object can be out-zoomed. As a result, every pictogram inside `BookCo` will be invisible.

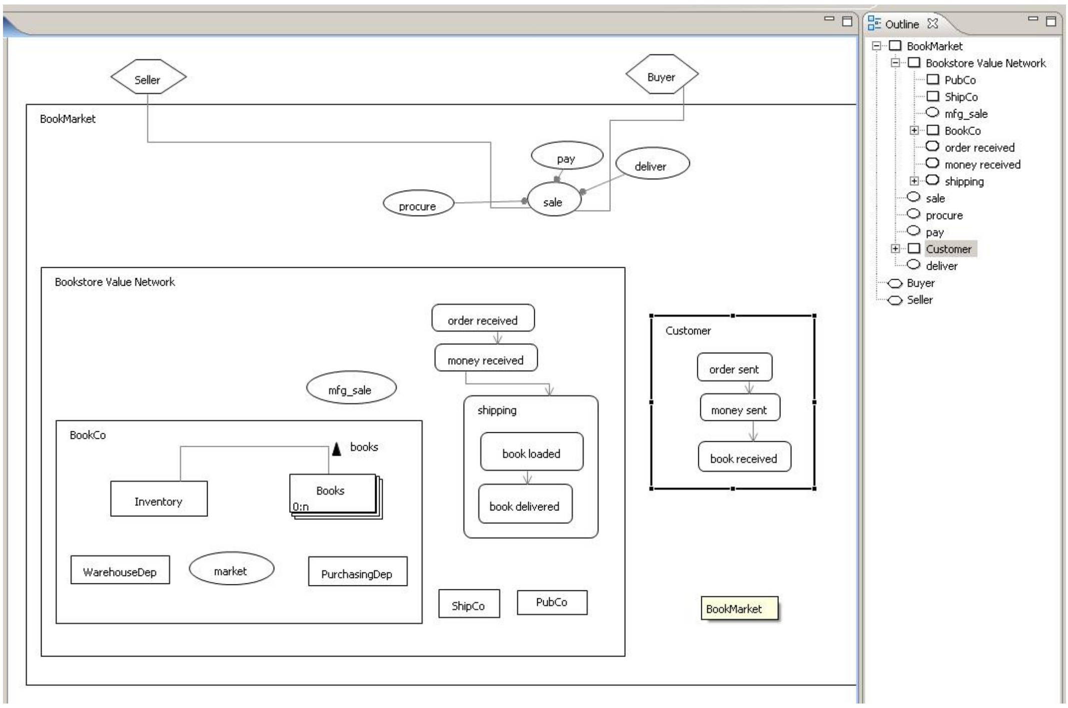


Figure 6. Model of the online bookstore created in the Adora modeling tool

Adora does not differentiate black-box and white-box of an object. In Adora, black-box and white-box are actually mixed in the same representation of the object. The scenarios and component objects constitute the white-box whereas the states and property-like objects (e.g. `Inventory` and `Books`) make up the black-box of the object being mentioned. The white-box of behavior can be understood as the way scenarios are extended from a given scenario (for example, the scenarios `order`, `pay` and `deliver` extend `sale`).

2.3.2. KobrA

KobrA defines a recursive approach for developing component-based software system [19]. Each component should be represented by sets of artifacts: specification and realization. Both of them have structural and behavioral diagrams of a component. The main difference is the specification describes the component as a black-box whereas the realization shows the component as a white-box. As illustrated in Figure 7, the specification consists of structural model (typically represented using UML class diagram), behavioral model (typically represented using UML statechart diagram) and other models. The realization has structural model (typically represented using UML class diagram), activity model (typically represented using UML activity diagram) and other models [12].

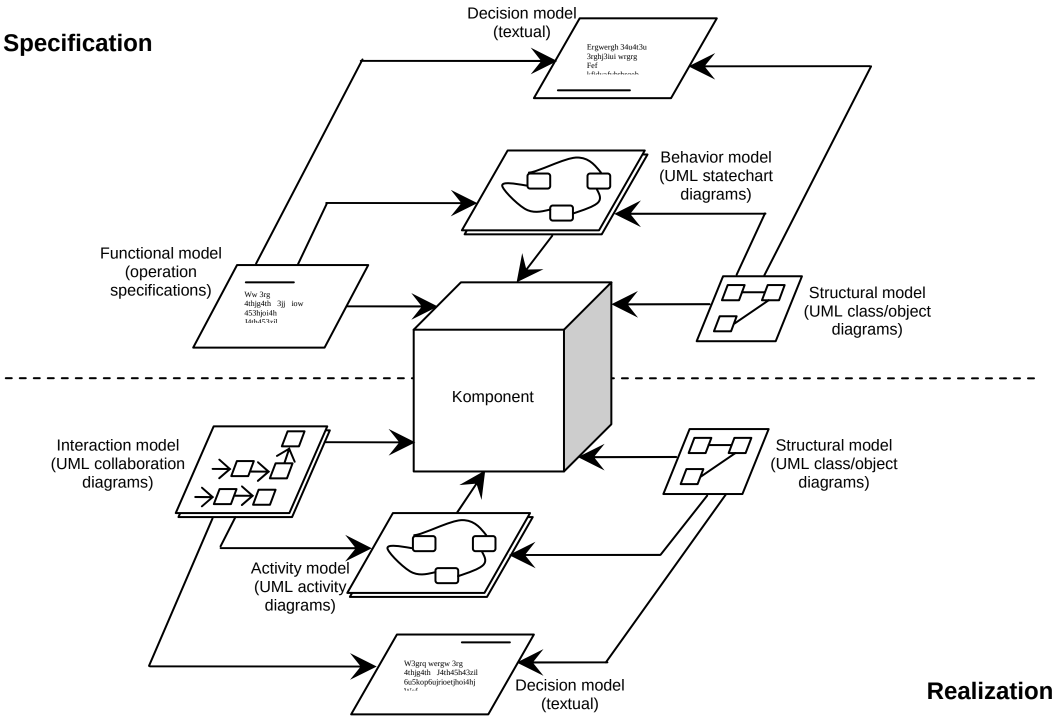
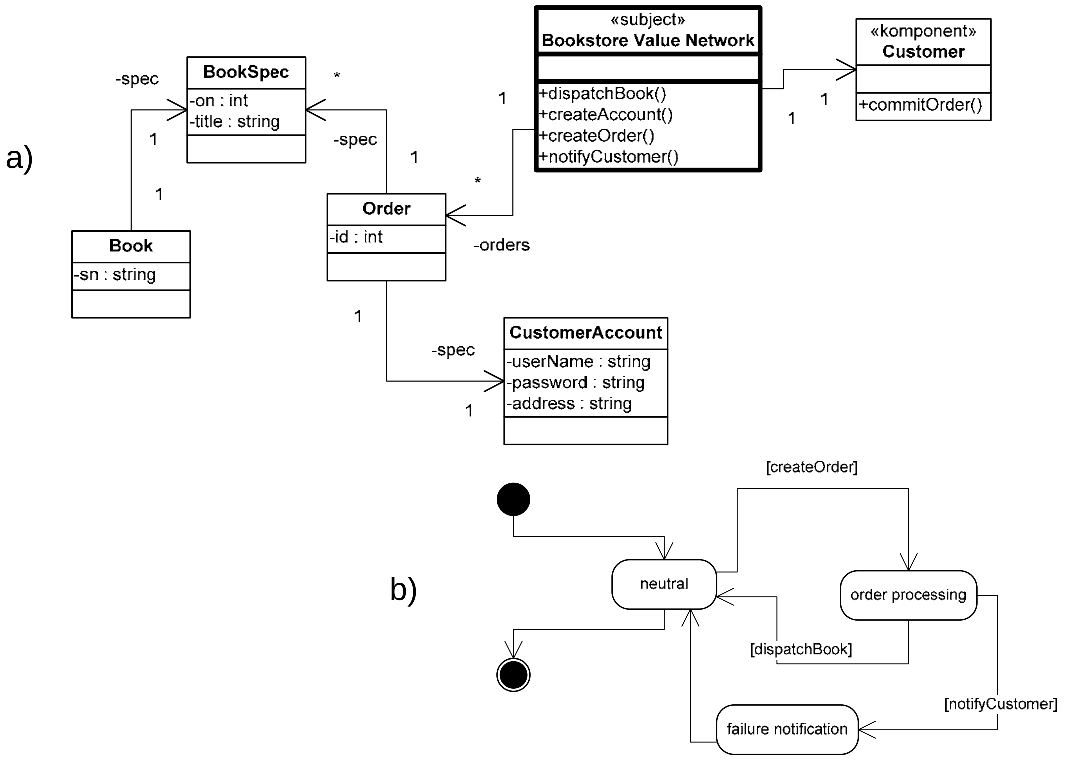


Figure 7. Specification and Realization of Komponent in KobrA [12]

Figure 8 shows the specification for the component Bookstore Value Network. The specification of this component consists of two diagrams. Figure 8 a) is a class diagram that shows the component (its pictogram is rendered in bold) and main concepts to represent the customer, the order, the book ordered and the book description in the catalog. This diagram can be considered as a black-box representation of the component Bookstore Value Network because it does not reveal any components that constitute the Bookstore Value Network. Figure 8 b) is a statechart diagram that is given to describe the behavior of the black-box of Bookstore Value Network: doing nothing, processing order and notifying some failure in case the order cannot be correctly completed. Note that the operations of Bookstore Value Network declared in the class diagram should be represented in the transitions of the state-chart diagram.

One of the operations of Bookstore Value Network is createOrder. It takes the identification number of the book to be sold and necessary information about the buyer to create an instance of order that would be processed later. In KobrA, this operation can be described as given in Figure 8 c).



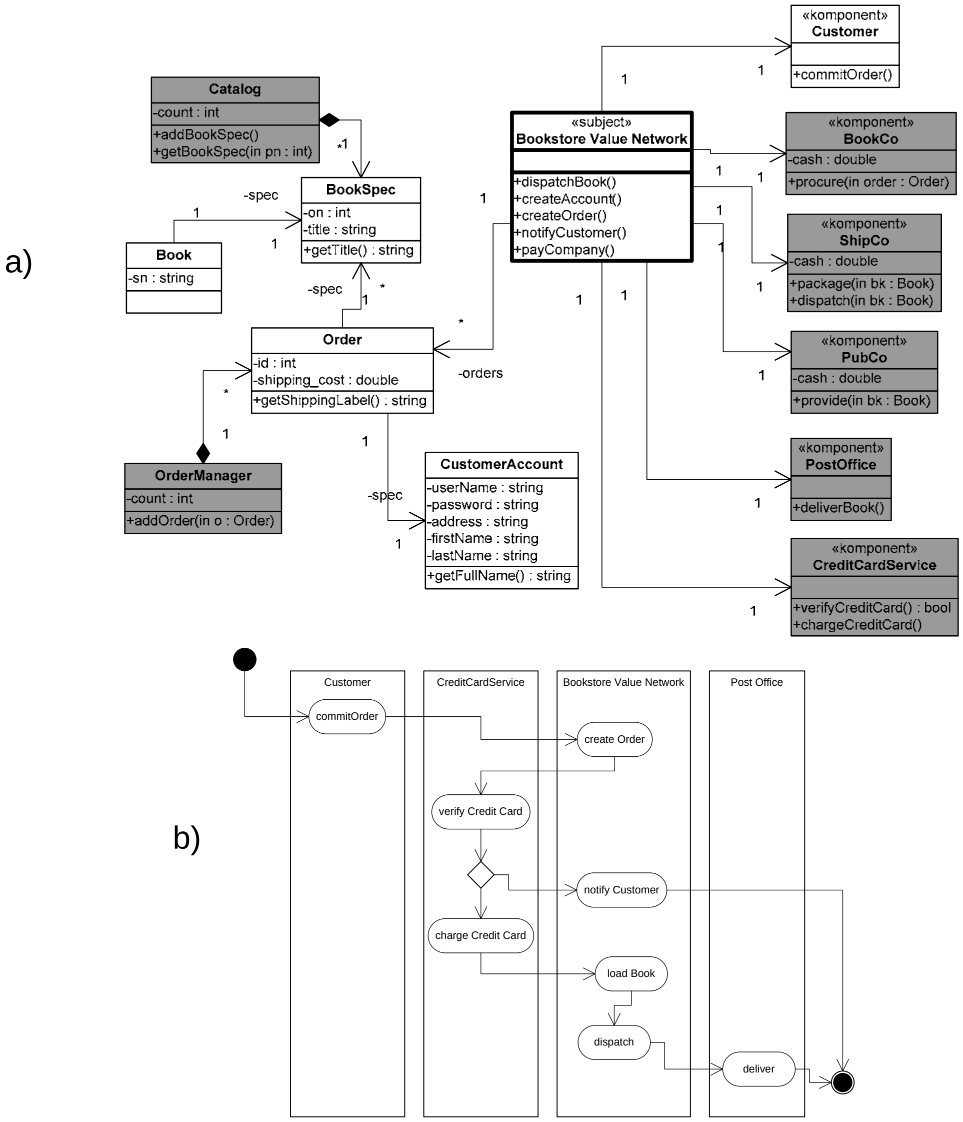
c)

Name	createOrder
Description	Once the customer has committed, an order is created and ready to be processed
Receives	PN of the book to buy, customer's shipping address
Returns	A newly-created order
Reads	An instance of BookSpec and an instance of CustomerAccount
Assumes	input PN represents some book managed by the Bookstore Value Network
Result	An order is created that binds input PN and the ID, address of the customer

Figure 8. Komponent specification of the `Bookstore Value Network`

Figure 9 illustrates the realization of the component `Bookstore Value Network`. Figure 9 a) is a class diagram that is actually elaborated from the class diagram of the specification (see Figure 8 a)) by adding more classes (filled in gray). Figure 9 b) is an activity diagram is also elaborated from what is depicted in the statechart diagram of the specification (see Figure 8 b)). This diagram describes what the component `Bookstore Value Network` and other components that represent the customer, the post office and the credit card service should perform to realize the sale.

Like the specification, operations defined in the realization need to be described in details. Figure 9 c) gives detailed description for the operation `payCompany` of the component `Bookstore Value Network`.



c)

Name	payCompany
Description	Once the book has correctly been delivered to the customer, ShipCo and BookCo are paid.
Changes	Cash of ShipCo and cash of BookCo
Reads	shiping_cost of Order
Assumes	The book has correctly been delivered to the customer
Result	The cash of ShipCo is increased by a predefined shipping cost (field shiping_cost of class Order). The cash of BookCo is increased by the book price minus the predefined shipping cost.

Figure 9. Komponent realization of the Bookstore Value Network

In KobrA, the specification and the realization of a component correspond to the black-box and the white-box representation. The state of a component can be shown as black-box and white-box too (any state if the specification can be composite). The component containment can be regarded as the organizational hierarchy. Black-box of behavior can be seen in the statechart diagrams. White-box of behavior can be observed in activity diagrams. Model coherence in KobrA is maintained by the intra-diagram and inter-diagram rules that are informally stated. It is up to the modelers to keep their model consistent by following these rules.

2.3.3. OPM

Object Process Methodology (OPM) defines object, process, state and link as its building blocks [13]. Object and process can be physical or informatical. Physical objects are tangible whereas informatical objects can reside in short-term media like information system, human mind... The composition of object and process can be represented using aggregation link. Informatical objects can represent a physical object as a black-box via exhibition link. Objects participate in process via agent link and instrument link. Figure 10 summarizes building blocks of OPM.

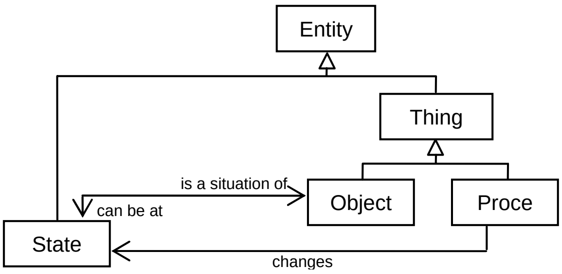


Figure 10. Building blocks of OPM [13]

OPM is supported by a tool called OpCat. Using this tool, the modeler can create her model that consists of diagrams and elements. Each element in a diagram can be in-zoomed and out-zoomed. This zooming operation may result in creating an additional OPM diagram. Figure 11 illustrates a diagram created in OpCat for the bookstore model. `BookMarket`, `Bookstore Value Network`, `Customer`, `BookCo`, `ShipCo`, `PubCo` are physical objects. `Inventory`, `Cash` and `Catalog` are informatical objects that characterize the physical object `Bookstore Value Network`. Object `Bookstore Value Network` is also characterized by the informatical process `Sell`. Similarly, `WantedPN`, `Cash` and `Bookshelf` are informatical objects that, together with informatical process `Buy`, characterize the physical object `Customer`. This characterization is expressed via exhibition relations that is shown using lines with triangles having a smaller blackened triangle inside. Physical process `sale` represents the interaction between the two physical objects `Bookstore Value Network` and `Customer`. This process changes the `Cash` and the `Catalog` of the `Bookstore Value Network`. On the side of the `Customer`, its changes the `Cash` and the `Catalog` but takes the informatical object `WantedPN` as input.

Aggregation relations (lines with blackened triangles) are used for specifying the organizational hierarchy among physical objects and the behavioral hierarchy among processes. The process `sale` is broken down into `order`, `pay` and `deliver`. On the side of the `Bookstore Value Network`, process `sale` is broken down into `receiveOrder`, `credit` and `sendBook`. On the side of the `Customer`, process `Buy` is broken down into `commitOrder`, `debit` and `receiveBook`. Note that an informatical object called `Order` is an output of `commitOrder` but an input of `receiveOrder`.

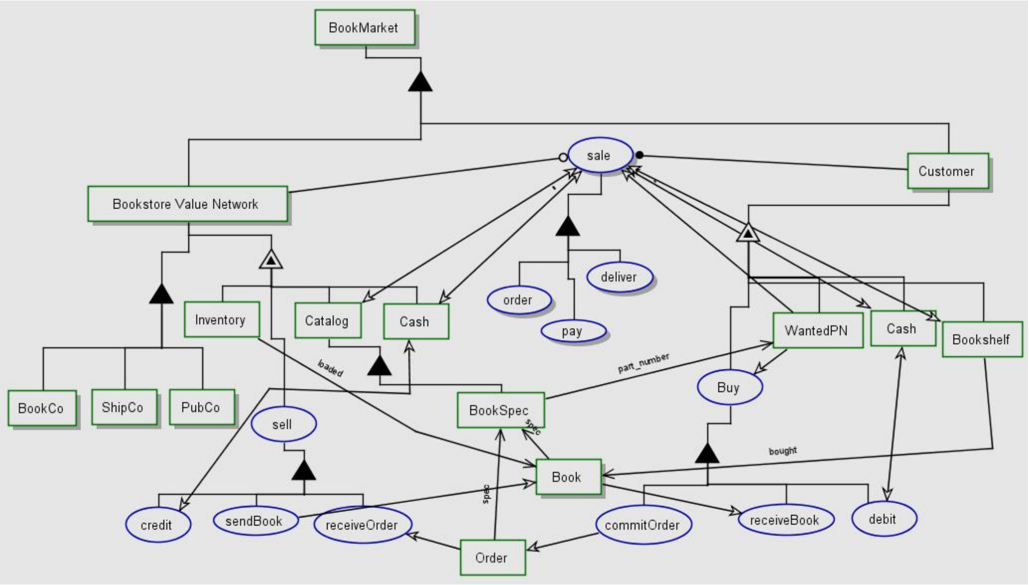


Figure 11. The model of the online bookstore created in OpCat

In OPM, the notion of black-box and white-box can be applied to both object and process. The black-box representation of an object is its exhibition including information objects and informatical processes. The white-box representation of an object is its aggregation of other physical objects. The white-box of a process can be considered as aggregation of other processes. The organizational hierarchy is expressed via consecutive object aggregations. The behavioral hierarchy can be understood as consecutive process aggregations. In the tool OpCat, an initial OPM diagram is made when the model is created. When an object or process is in-zoomed, a new diagram is created if this object or process is in-zoomed for the first times. It is up to the modeler to keep the newly-created diagram to existing ones. The coherence of model is therefore maintained by the modeler.

2.3.4. Comparison between AdorA, KobrA, OPM and SeamCAD

In the SeamCAD modeling language, the notion of black-box and white-box can be applied to working objects, properties and behavior. The black-box of a working object is represented in terms of its properties and localized actions. The white-box of a working object is expressed in terms of working objects and distributed actions. For a model element that is a property, a distributed action or a localized action, the black-box is the model element itself and the white-box is its component model elements that must be of the same kind. The way a working object is broken down into component working objects is consistently carried out from the top working objects down to the leaf ones forms the organizational hierarchy. Similarly, the way a model element that is either a property, a distributed action or a localized action is broken down into component model elements of the same kind is consistently carried out from the top one down to the leaf ones forms the functional hierarchy. The well-formedness of model in the SeamCAD modeling language is maintained by the computer-aided tool. This aspect will be elaborated in Chapter 4.

Having analyzed OPM, Adora and KobrA in the previous sections, we compare them to the SeamCAD modeling language (see Table 3) in a grid that covers not only the four hypotheses used in evaluating modeling methods in the beginning of this chapter but also widely-known modeling issues such as black-box, white-box, state modeling and notation. Specifically, the grids consists of the following criteria

- whether the method can distinguish the black-box from the white-box
- how systems can be described as black-box
- how systems can be described as white-box
- how the organizational hierarchy is described
- how the state can be represented
- black-box of behavior
- white-box of behavior
- behavioral hierarchy within the black-box of system
- behavioral hierarchy within the white-box of system
- model coherence and consistency
- notation

Table 3. Modeling aspects of SeamCAD, Adora, KobrA and OPM

Modeling Aspect	**Adora**	**KobrA**	**OPM**	**SeamCAD**
Black-box / White-box applicability	None Black-box and White-box are mixed	Komponent State	Object Process	Working object Property Localized action Distributed action
Black-box of system	States of object. Black-box and White-box are mixed	Specification of Komponent Structural diagrams, behavioral diagrams, functional…	Exhibition of object Informatical objects & informatical processes	Working Object as Whole Properties and localized actions
White-box of system	Nested objects Black-box and White-box are mixed	Realization of Komponent Structural diagrams, behavioral diagrams, functional…	Aggregation of Object	Working Object as Composite Component working objects and distributed actions
Organizational Hierarchy	Nested Objects	Komponent Containment	Aggregation of Object	Working Objects as Composite
State modeling	State	State	Informatical Object	Property
Black-box of behavior	Scenario	Action	Process Can be in-zoomed in OpCat	Action as Whole
White-box of behavior	Scenario Connection A scenario can be connected to component scenarios	Activity	Aggregation of processes Can be out-zoomed in OpCat	Action as Composite
Behavioral hierarchy in the black-box of system	Connected Scenarios Black-box and White-box are mixed	Composite state	Aggregation of Process that characterize an Object	Localized Action as Composite
Behavioral hierarchy in the white-box of system	Connected Scenarios Black-box and White-box are mixed	Activity	None	Distributed Action as Composite
Model well-formedness	Diagrams are rendered as partial representations of a coherent model	Rules (intra-diagram, inter-diagram, containment…)	- Additional diagrams are created based on the root diagram - Diagram consistency is up to the modeler	Diagrams are rendered as partial representations of a common enterprise model
Notation	UML-like	UML	OPM notation	SeamCAD notation

Chapter 3: The SeamCAD Modeling Language

***Overview**: We describe the SeamCAD modeling language in this chapter. First, the foundations of the modeling language are presented, including the SEAM method, the Living Systems Theory and the Reference-Model of Open Distributed Processing. Then the building blocks of the modeling language are informally defined in English. This informal definition is followed by a meta-model that consists of a diagram expressing the informally-defined building blocks and a list of well-formedness rules that must be respected by any enterprise model instantiated from the meta-model. The definition of the SeamCAD language is exemplified by an enterprise model of an online bookstore. Next, the notation used in the SeamCAD modeling language is defined. Last but not least, the meta-model is rigorously formalized in Alloy - a lightweight declarative language based on the set theory and the first-order logic.*

In this chapter, we first describe the foundation of the SeamCAD modeling language. Then, we define the SeamCAD modeling language in three steps: informal definition, a more formal definition and finally formalization. In each step, the example of the online bookstore is used to illustrate the definition presented.

3.1. Foundations

The SeamCAD modeling language takes the Living Systems Theory, the Reference-Model of Open Distributed Processing and the SEAM method as its foundations.

SEAM Method

SEAM is a method developed at our research group for modeling EA [3] [4] [5]. By the time this Ph.D. work got started, the Living Systems Theory and the RM-ODP were used by SEAM to define a vocabulary for modeling EA in a hierarchical way. This vocabulary and the approach of representing EA hierarchically served as a basis for several Ph.D. projects at that time including this dissertation. It was our goal for this Ph.D. to develop a modeling framework based on the initiative of the SEAM modeling vocabulary and spirit. To achieve this goal, we rigorously define the modeling terms by means a meta-model, which is then formalized using a language that is capable of processing the first-order logic. The notation of these modeling terms is also defined. This notation definition includes the notation rules that specify the way notation pictograms are put together in a diagram. Thanks to these rigorous definitions, it was feasible to develop a computer-aided tool that specifically implements the defined modeling terms. In summary, the main results of this Ph.D. are the definition of a modeling language and the development of a computer-aided tool that are altogether called SeamCAD. In other words, SeamCAD consolidates the modeling vocabulary and spirit that were initiated in the SEAM method.

Living Systems Theory

James Greer Miller introduced the concept of level in [7]. He made a thorough cross-discipline analysis and synthesis of the functions and behavior of living systems. He published his results in 1978 (first edition) and in 1995 (second edition). His theory is called the “General Theory of Living Systems” or “Living Systems Theory” (LST). To develop his theory Miller analyzed 4000 publications from multiple living systems disciplines. He then developed a model that can be used to reason about any living system (from individual cells to supranational organizations such as the United Nations Organization).

One of the most important concepts is that of *level*. According to Miller “…the universe contains a hierarchy of systems, each more advanced or ‘higher’ level made of systems of the lower levels”. He identifies seven distinct levels for living systems: cells (free-living cells and aggregated cells), organs, organisms (such as humans…), group (such as families, workgroups…), organization (such as commercial companies…), society (such as countries) and supra-national systems (such as inter-governmental organizations …). This level distinction is tightly linked to people’s experience in perceiving and studying the world of livings systems. Depending on the goal of the modeler, it is possible to have more or less levels.

To be able to model enterprises, the language should define the model not only confined to IT and software-intensive systems but also to business-related systems. In SeamCAD, the enterprise model is structured in LST-inspired levels that are called *organizational levels*. Making enterprise models hierarchical is also a convenient way to structure the enterprise models as people tend to reason in terms of hierarchy [2].

Reference Model for Open Distributed Processing (RM-ODP)

Within the organizational levels, we use RM-ODP [8] to represent what is perceived. RM-ODP is a standard that defines the concepts necessary to build “distributed information processing services to be realized in an environment of heterogeneous IT resources”. RM-ODP also proves to be suitable for general modeling.

The Reference Model for Open Distributed Processing (RM-ODP) is an ISO/ITU standard. RM-ODP defines a modeling infrastructure for distributed IT systems within organizations. The RM-ODP standard is composed of four parts. Part 1 is an overview of RM-ODP and is non-normative. Part 2 defines the fundamental concepts needed for modeling Open Distributed Processing systems. Part 3 presents an application of Part 2 for particular viewpoint specification languages (i.e. enterprise, information, computational, engineering, technology viewpoints). Part 4 is a partial formalization of the previous parts.

RM-ODP is known especially for its Part 3 that defines requirements for viewpoint languages useful to describe an IT system and its environment [15] [20]. For example, the enterprise viewpoint is useful for describing the enterprise in which the IT system will be deployed; the information viewpoint is useful for describing the IT system specification; the computational viewpoint is useful for describing the computing structure of the IT system; the engineering and technology viewpoints are useful for the implementation of the IT system. All these viewpoints refer to the terminology defined in RM-ODP Part 2 (e.g. object, state, action, activity, type, and instance).

3.2. Informal Definition

In this section, the SeamCAD modeling language is presented informally. This informal definition of the modeling language [21] is followed by explanation of an enterprise model of the online bookstore using the defined building blocks of the SeamCAD modeling language.

3.2.1. Building blocks of the SeamCAD modeling language

According to RM-ODP part 2 (i.e. the foundations), an entity is any concrete or abstract thing of interest in the universe of discourse. An entity can be considered as atomic or as non-atomic (i.e. composed of parts of the same kind). An entity is represented in the model as a model element. So, a model element can be seen as whole or as composite. A system may be referred to as an entity. A part of a system may itself be a system, in which case it may be called a subsystem. The model element that corresponds to a system is an object. An object can be seen as whole (i.e. this corresponds to the external view of the object, also called model-based specification) or as composite (i.e. this corresponds to the internal view, or implementation, of the object). Other kinds of entities can be modeled as action and state. Action and state can also be seen as whole or as composite. An action (state) seen as composite can be broken down into component actions (states); these component actions (states) can be further broken down into smaller component actions (states). This hierarchy of actions (states) corresponds to the *functional level hierarchy*. The functional level hierarchy includes *functional levels*. It is orthogonal to the organizational level hierarchy defined in the previous section (in which an object is broken down into its component objects). In each organizational level, we can find multiple functional levels.

According to RM-ODP part 3 (i.e. the architecture), a system specification has five viewpoints: enterprise, information, computational, engineering and technology viewpoint. In the context of hierarchical systems, we model organizational levels (cf. Miller) made of *working objects* (i.e. an object that represents a system). The working objects can be specified by either an *information specification* or a *computational specification*. An information specification represents a working object as whole (or seen from outside). It consists of *properties* and *localized actions*. The properties represent the states of the working objects. The localized actions model the working object's responsibility. A specification viewpoint represents a working object as composite (or seen from inside). It consists of working objects and actions happening between them that we call *distributed actions*. Note that as RM-ODP part 2 just defines the term action, we have to define the localized actions (for the information specification) and the distributed action (for the computational specification).

Organizational Level in Terms of Working Objects as Whole / Composite

A computational object seen as whole contains properties that represent either the fact that a localized action executes (called transaction) or elements of "knowledge" (called concepts) or parameters exchanged with the environment (called parameters). These concepts describe "knowledge" of the working object about itself or about other working objects belonging to the same organizational level. The distributed action is described with pre-conditions and post-conditions in terms of the participating working objects as whole. Once a working object is seen as composite, it contains component working objects and distributed actions. By changing the view of the working object from the whole to the composite, the modeler descends to the next organizational level.

Functional Level in Terms of Distributed Actions, Localized Actions and Properties as Whole / Composite

Not only working objects but also distributed action, localized action and property can be treated as whole and as composite. Once the modeler breaks down a distributed action, she descends to the subsequent functional level. For the sake of level consistency, all properties and localized actions that are semantically bound to this distributed action should always be at the same level of granularity. They must be broken down accordingly when she changes to subsequent functional level. In the opposite direction, when she gets back to the precedent functional level by imploding a distributed action, these properties and localized actions are imploded as well. This principle is crucial for navigating in the model as it prevents properties and actions from being represented at different level of granularity, which potentially makes the modeler confused even for the very part of model created by her.

Relation

The expressiveness of the enterprise model created in the SeamCAD modeling language can be enriched by putting model elements in relation. There are three kinds of relation that are intrinsic to the SeamCAD modeling language: i) between a model element and its component model elements; ii) between a distributed action and localized actions and properties that are bound to it; iii) between a localized action and a distributed action that implements it. The first is called composition, the second goal binding and the third means binding. Between a model element and each of its component elements, there is a composition. For each distributed action, there could be multiple participating working objects. In each of these working objects seen as whole, there are essentially a localized action and its (stateless) transaction that represent the responsibility of this working object when taking part in the distributed action. There could be other (stateful) properties and localized actions that additionally represent this responsibility. A goal binding relates any of these properties or localized actions to the distributed action being mentioned. On the other hand, the means binding relates a localized action of a working object seen as whole and a distributed action of the same working object seen as composite that implements the localized action. This relation can be understood in the sense that a distributed action of the working object as composite represents collaboration between component working objects. The component working objects and the way they collaborate with one another should implement some responsibility of the working object, which is represented by some localized action defined the in same working object seen as whole.

There are other kinds of relation in the SeamCAD modeling language that are actually borrowed from the Unified Modeling Language. An association can be made between two different properties of the same working object or from a property to itself. For instance, in the model of the online bookstore, an order is associated with a specification of the book being ordered. An association may have role names and cardinalities at the two ends.

A transition is a directed relation that connects two actions or connects a single action to itself. Transitions represent sequence constraints for the behavior of a working object. From a localized action, there could be several outgoing transition and several incoming transitions. A condition can be attached to each transition to determine how the transition is fired. There are two special transitions: start transition and stop transition. The former indicates that the action being connected can take place just after the beginning of the behavior of the working object where it is defined. The latter implies that the action being connected can take place before the end of the behavior of the working object where it is defined. Like normal transitions, a condition can also be attached to any of these special transitions. Note that if two different actions are connected by a transition, they must be defined in the same working object.

A generalization is used for specifying an is-a relationship among model elements of the same kind. Most frequently, generalization is exploited to describe a generic property or working object from which concrete property or working object are specialized. In the model of the bookstore, although not represented, it is possible to have a working object that describes a generic company from which `BookCo`, `ShipCo` and `PubCo` are specialized.

A participation link is an undirected relation that specifies which working object takes part in which distributed action. A participation link determines the goal binding between the distributed action being connected and the corresponding transaction and the localized action of the working object being connected.

Table 4 summarizes the informal definition of the SeamCAD modeling language. The column to the left lists the building blocks. The column to the right lists the kinds of the building blocks. The column in the middle is where we can find the verbal description of the building blocks.

Table 4. Informal definition of the SeamCAD modeling language

Building block		Informal definition	Kind
Working Object		Represents any business unit, IT component or software component of the enterprise.	Hierarchical element
Property	Stateful property	Externally-observable properties that characterize a given Working Object seen as whole	Hierarchical element
	Stateless Property (Transaction)	Representation of the occurrence of a localized action. Transaction is also considered as a context in which normal properties are defined	Hierarchical element
Localized Action		Externally-observable actions performed by a given Working Object seen as whole. It also represents a service offered by the working object.	Hierarchical element
Distributed Action		Interaction between multiple working objects that are distributed into localized actions of participating Working Objects seen as composite.	Hierarchical element
Association		Relation between two properties of the same working object.	Expressive relation
Generalization		Relation between two properties or two working objects	Expressive relation
Transition		Relation between two localized actions that are parented by the same localized action	Expressive relation
Start / Stop Transition		Relation coming to for going from a localized action	Expressive relation
Participation		Relation between a working object and a distributed action in which it participates.	Expressive relation
Composition		Relation between a model element and its parent model element.	Intrinsic relation
Goal binding		Binding of a property or a localized action to a distributed action that is at the same functional level.	Intrinsic relation
Means binding		Binding of a distributed action to a localized action that it implements. The two actions must be hosted by the same working object	Intrinsic relation

To explain the actions and how they are related by goal/means bindings, Figure 12 gives some illustration [22]. The stereotypes used for model elements that are visible in the diagrams of Figure 12 make explicit from which building blocks they are instantiated. In Figure 12 a), working object S consists of two component working objects: `AinS` and `BinS`. They participate in a distributed action `M_S`. The working object `AinS` has a localized action called `M_A` that represents the responsibility of `AinS` in taking part in `M_S`. It also represents the main service of `AinS`. Similarly, the working object `BinS` has a localized action called `M_B` that represents the responsibility of `BinS` in taking part in `M_S`. It also represents the main service of `BinS`. There are two goal bindings in Figure 12 a): from `M_A` to `M_S` and `M_B` from to `M_S`. Note that only `S` is seen composite in Figure 12 a). The other elements are seen as whole in this diagram.

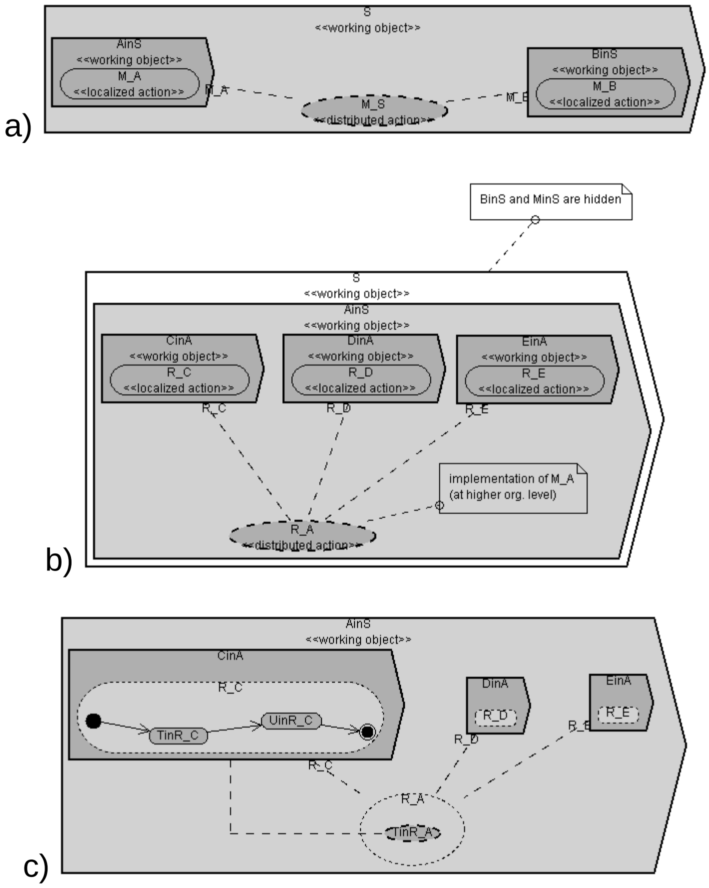


Figure 12. Examples of actions and bindings

In Figure 12 b), the working object `AinS` is seen as composite whereas `BinS` and `M_S` are hidden. `AinS` has three component working objects, namely `CinA`, `DinA` and `EinA`. The distributed action between them, `R_A`, implements the localized action `M_A` that is visible in Figure 12 a). There is a means binding from `R_A` to `M_A`, which can be made explicit by a note attached to `R_A` as we can see in Figure 12 b). In this diagram, there are three goal bindings: the goal binding from `R_C` to `R_A`, the goal binding from `R_D` to `R_A` and the goal binding from `R_E` to `R_A`.

In Figure 12 c), the working object `S` is hidden. The distributed action `R_A` and the three localized actions that are bound to it via goal bindings are seen as composite. Being seen as composite, the localized action `R_C` looks like an activity. It has two component localized actions: `TinR_C` and `UinR_C`. It has three transitions: the start transition running to `TinR_C`, the transition between `TinR_C` and `UinR_C`, the stop transition running from `UinR_C`. Note that there is also a goal binding between the localized action `TinR_C` and the component distributed action `TinR_A` of `R_A`.

Relationship to RM-ODP

A total of five viewpoints are defined in RM-ODP. They are enterprise viewpoint, information viewpoint, computational viewpoint, engineering viewpoint and technology viewpoint. These viewpoints describe the different aspects necessary to model an IT system. Each viewpoint has its own modeling language. In our approach, the goal is to have the same modeling language regardless of the subject to be modeled (e.g. business entity or IT system) and to have a relatively small set of heuristics for the specific aspects of each subject. Hence, we base our work directly on RM-ODP Part 2 and we systematically use the concepts defined in RM-ODP Part 2 to represent systems that span business and IT.

Our approach is original because it does not rely on the RM-ODP viewpoints [23] [22]. These viewpoints describe the different aspects necessary to model an IT system. Each viewpoint has its own modeling language. In our approach, the goal is to have the same modeling language regardless of the subject to be modeled (e.g. business entities or IT systems) and to have a relatively small set of heuristics for the specific aspects of each subject.

RM-ODP Part 2 first defines the *basic interpretation concepts*. These concepts are necessary to relate the universe of discourse to the model and to define the model elements. RM-ODP Part 2 then defines the *basic modeling concepts* (e.g. object, action, and activity) and the *specification concepts* (e.g. type, instance). These are the concepts necessary to fully specify the model elements.

RM-ODP Part 2 defines basic modeling concepts such as object, action, state. To directly support system modeling with RM-ODP Part 2, we had to define a few more concepts than those in the standard. We present these concepts in this section.

Our goal is to model systems. As defined in RM-ODP a system is *something of interest* seen *as a whole or as comprised of parts.* We consider the concept of system as an agreed conceptualization between the modelers. We define the *working object* as the model element that corresponds to the system conceptualization. *Working object* is a specialization of the concept of *object* defined in RM-ODP: The original *object* is not associated with the system conceptualization. In the example of the online bookstore, `Bookstore Value Network`, `Customer`, `BookCoMarket`, `BookCo`, `PubCo` and `ShipCo` are all working objects. This means that they are all perceived as systems in the universe of discourse.

We have also refined the definition of the different kinds of actions. In RM-ODP, actions are divided into internal actions and interactions. In RM-ODP Part 2, it is written: *The set of actions associated with an object is partitioned into internal actions and interactions.* To model systems as we propose, we need two kinds of interactions: distributed action and localized action. A *localized action* is an action of one working object of interest (represented as whole) and involves one or more working objects in its environment. A *distributed action* is an action of one working object of interest (represented as composite) and involves one or more of its component working objects and it may or may not involve working objects in the environment of the working object of interest.

Finally, we have introduced concepts necessary to structure the state space. RM-ODP Part 2 defines the concept of state as, *at a given instant in time, the condition of an object that determines the set of all sequence of actions in which the object can take part*. Our goal is to describe the state at the same level of detail as the behavior. For this reason, it was important to add a means to structure the state. This is the concept of property. Properties can be stateless or stateful. Stateless properties represent occurrences of actions. Stateless properties are called *transactions*. They are similar to the stateless objects presented in [24]. One special transaction is the lifecycle transaction that represents the overall working object lifecycle. Transactions are useful for representing the context in which stateful properties exist. Stateful properties store the system's state. They are similar to UML attributes except that they can be hierarchical (properties can be composite as well). Global properties exist in the context of the system lifecycle. They are created at the system's initialization and disappear at the system's termination. Local properties exist in the context of transactions with a shorter lifespan than the lifecycle transaction.

In summary, RM-ODP Part 2 is well adapted as ontology for enterprise models but it would require a few extensions to be perfectly suitable. The extension we propose consists of: the way we model systems with objects, the definition of the distributed actions and localized actions and the concept of properties to structure the state.

3.2.2. Explanation of the enterpise model of the online bookstore

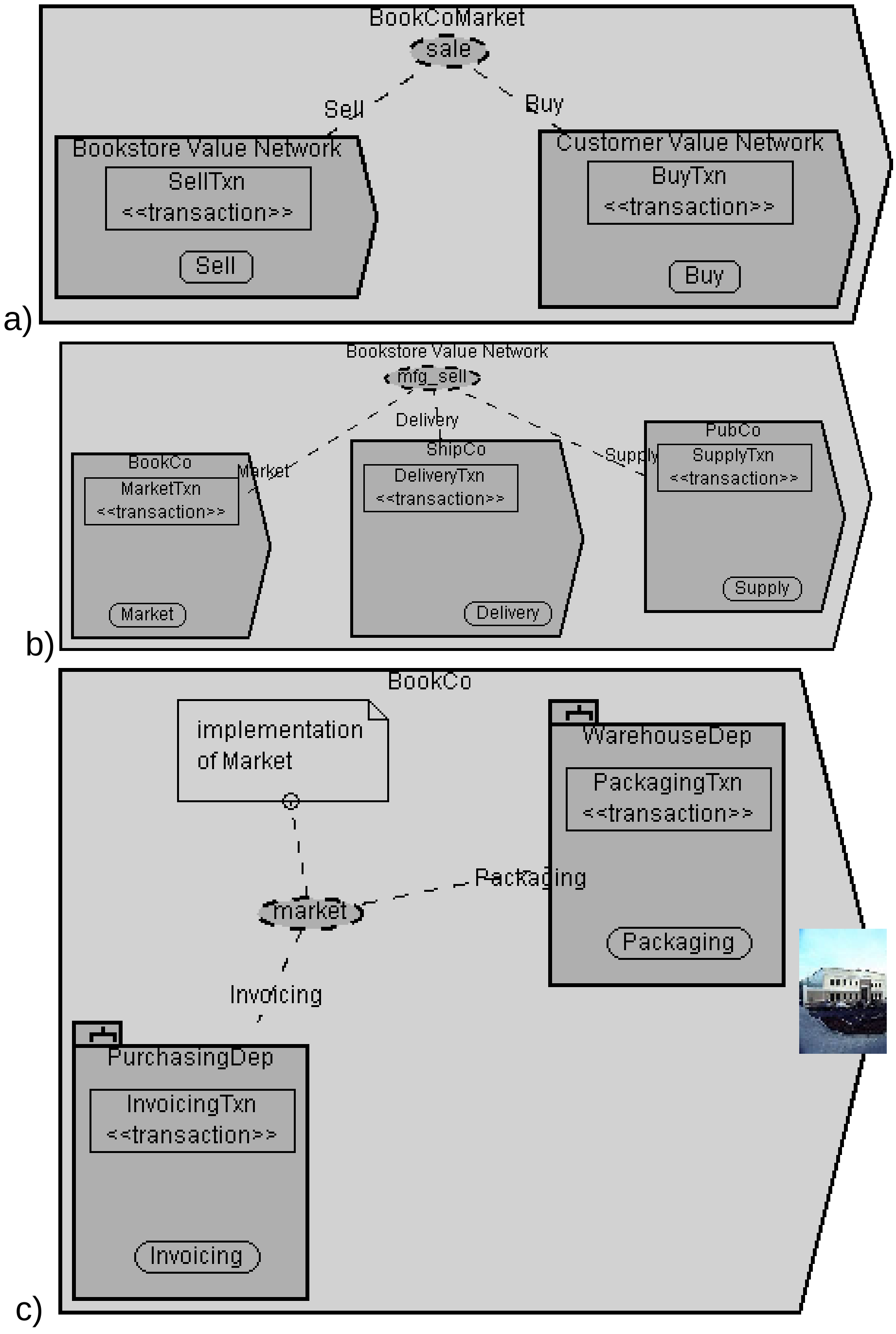
In this subsection, we illustrate the SeamCAD modeling language by explaining the details of the enterprise model of the online bookstore.

Figure 13 has a total of 5 diagrams that shows different organizational levels of the bookstore model. As illustrated in Figure 13 a), at the top organizational level, BookCoMarket as composite consists of two value networks: Bookstore Value Network and Customer Value Network. They are all working objects. Members of Customer Value Network place orders at the Bookstore Value Network, make necessary payment and wait for the delivery of the books they ordered. The overall collaboration between the two value networks is called distributed action sale. The Bookstore Value Network performs the localized action Sell, which represents the responsibility of Bookstore Value Network while taking part in the distributed action sale. The Customer Value Network performs the localized action Buy - responsibility of Customer Value Network while taking part in the distributed action sale. In these two value networks, SellTxn and BuyTxn represent the occurrences of the localized action Sell and Buy, respectively. Note that the two value networks are seen as whole in Figure 13 a).

In the second organizational level (Figure 13 b), Bookstore Value Network as composite is composed of there companies: BookCo, PubCo and ShipCo. They collaborate with one another through a distributed action mfg_sell. This distributed action implements localized action Sell of Bookstore Value Network seen as whole. It represents the entire collaboration of the three companies: order procurement, inventorial operation, invoicing and book delivery.

In the third organizational level (Figure 13 c), BookCo as composite is made up of two departments: PurchasingDep (responsible for identifying the book described in the customer's order and invoicing) and WarehouseDep (responsible for picking book from an inventory and packing it). Note that both of them are specified as whole when participating in distributed action market. The responsibilities of the two departments are represented under two localized actions called Invoicing and Packaging. The distributed action market is actually the implementation of localized action Market of BookCo in the previous organizational level. Figure 13 d) illustrates that the distributed action market seen as composite has two constituent actions: procure (processing order) and pack (book packaging) that all belong to the second functional level (whereas distributed action market as whole belongs to functional level 1).

Accordingly, `PurchasingDep` as whole is also specified in the second functional level to have localized action `Pick`, `Invoice` and `PickTxn`, `InvoiceTxn` (properties representing transactions of `Pick` and `Invoice`). The sequences between the two localized actions `Pick` and `Invoice` are specified via transitions: the action `Pick` occurs first, then the action `Invoice` is performed. This department also has property `BookCatalog` which can be broken down independently of any distributed action. `WarehouseDep` has two localized actions: `Pick` and `Pack`. Note that the localized actions and their transaction are bound to the corresponding distributed action. This binding is essential for rendering diagrams at different functional levels (the diagrams shown in Figure 13 c) and Figure 13 d), for instance) illustrating how the challenge *granularity* is addressed.



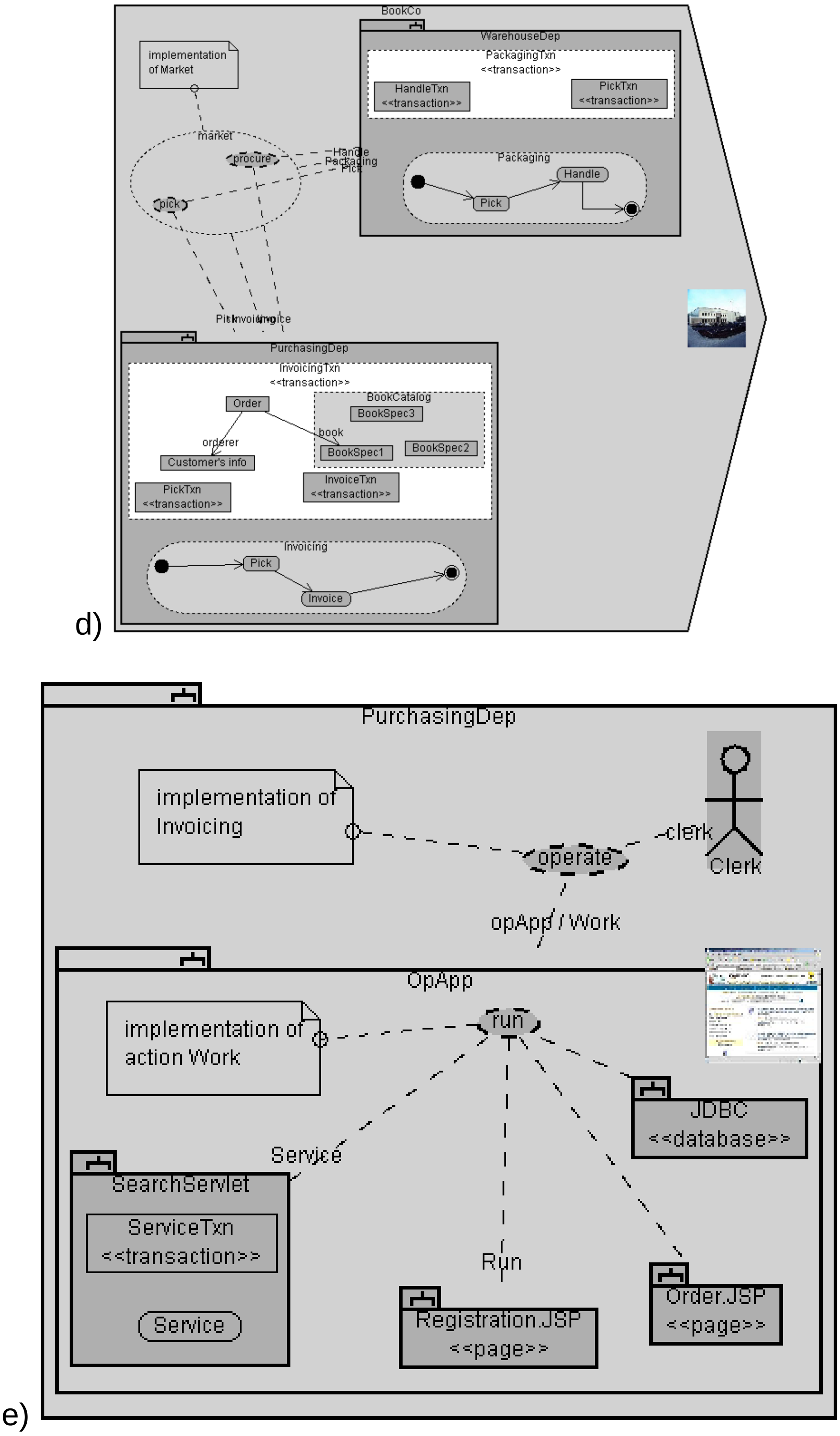


Figure 13. An enterprise model of the online bookstore in the SeamCAD modeling language

In the fourth organizational level, `PurchasingDep` as composite shows that a `clerk` operates an application called `OpApp` (Figure 13 e)). It is possible to have a fifth organizational level to show the architecture of `OpApp`. This application seen as composite has `SearchServlet` (a Java servlet responsible for searching a book), web page RegistrationPage.JSP (for customer registration) web page Order.JSP (order processing), and database interface JDBC. Note that the pictogram used for departments, the application and its components are different from that used for companies and value networks. This difference in pictograms can remind the modeler of the organizational level she is working: a business-related level or an IT-related one. A total of 5 organizational levels spanning market and IT implementation shows how the challenge *uniformness* is solved. The way models are managed by the tool in terms of a set of model elements (not as a set of weakly-related diagrams) linked together shows how the challenge *well-formedness* is addressed. The two value networks, three companies and two departments as well as the IT application can be specified as whole and as composite. This illustrates how the challenge *multi-entity* is addressed.

3.3. Formal Definition

In this section, the SeamCAD modeling language is defined in a more formal way than it was in the previous section. A meta-model that describing the building blocks and the well-formedness rules of the SeamCAD modeling language is presented followed by an explanation of the enterprise model of the online bookstore using in terms as an instantiation from the meta-model.

3.3.1. Meta-model

A meta-model is needed for modeling the building blocks of a modeling language. Typically, a meta-model consists of diagrams and rules. Diagrams visually show the building blocks of the modeling language and how they are related. They can be drawn using a widely-used diagrammatic language such as Unified Modeling Language (UML). Rules capture the well-formedness of the modeling language. They are also called well-formedness rules.

Figure 14 is a UML class diagram expressing building blocks of the SeamCAD modeling language [25]. The Working Object, Property and Action are all subtypes of a generic concept Hierarchical Element. This concept has a number of component elements and may have at most one parent element. The component elements and the parent element are expressed under association ends in this diagram: components and parent, respectively. Note that the concept Action is further specialized into the two kinds of action in SeamCAD: Distributed Action and Localized Action. All relations in SeamCAD are subtypes of another generic concept Relation. Most of the relations have a source element and a destination element, which represents the two model elements that they connect. The types of the source element and the destination element depend on the specific relation. For the Association, they are all Property. For the Participation, they are Working Object and Distributed Action. For the Action Transition, they are all Action. For the Generalization, they are all Hierarchical Element. The source element and the destination element are again expressed under association ends: source and destination, respectively. The only exception is on the Start Transition and the Stop Transition. They connect only one model element of the type Action.

A concept in the meta-model may have attributes. The most generic concept in the meta-model is called Element. It has two attributes of which data type is string: name and stereotype. As the concrete model elements and relations defined in the SeamCAD modeling language are (indirect) subtypes of this concept, they all have a name and a stereotype. The concept `Action` has two attributes of which data type is also string: `pre` and `post` that represent the pre-condition and the post-condition, respectively. As the two concepts that define the localized action and the distributed action in the meta-model are the subtypes of concept `Action`, each localized action or distributed action in SeamCAD has a pre-condition and a post-condition.

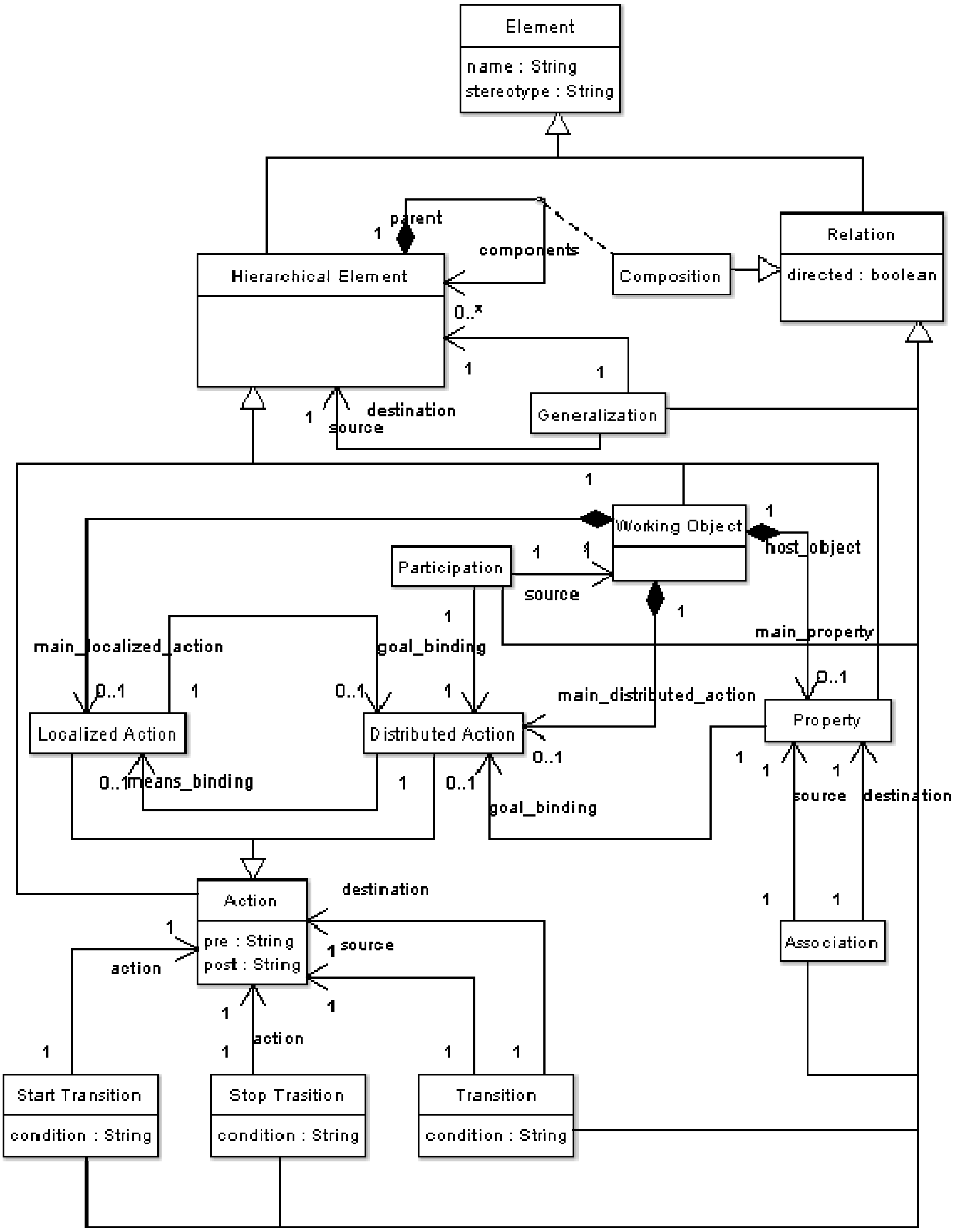


Figure 14. The UML class diagram that expresses the building blocks of the SeamCAD modeling language and how they are associated

The cardinalities shown in the diagram of Figure 14 indicate the number of instances of a specific building block that can be associated to an instance of another building block. Each working object has at most one distributed action (main_distributed_action), one localized action (main_localized_action) and one property (main_property). Any distributed action, localized action or property is associated to a working object where it is defined (host_object).

Each property or localized action is associated to at most one distributed action via goal binding (goal_binding). Each distributed action is associated to at most one localized action via means binding (means_binding).

Each expressive relation has exactly one source element and one destination element. However, it is possible for a model element to be connected by more than one relation. For instance, to express that a working object participates in two distributed actions, we need two instances of Participation both of which run from the working object. However, the two different instances of Participation run to different distributed actions.

As a modeling language, the SeamCAD modeling language should have well-formedness rules to ensure that the enterprise models created in this language are consistent and coherent. For example, there must be no cycle in the organizational level hierarchy. Another typical example is that, for a given model element, all of its component model elements must be of the same kind. Table 5 lists a total of 19 well-formedness rules that should be held for every model created using the SeamCAD modeling language. This table has three columns. The column to the left of this table gives enumeration names of the rules. The one in the middle presents informal descriptions of the rules. These descriptions are written in English. As the rules cannot be expressed in the class diagram given in Figure 14, the only way to relate them to this diagram is to list the diagrammatically-expressed concepts on which each rule matters. This is the purpose of the column to the right of Table 5.

In Chapter 1, we clarify that the work presented in this dissertation does not support model execution or model simulation. More concretely, the operational semantics of actions is not yet formally specified in the SeamCAD modeling language. This is in fact considered as future work of this thesis. As such, the well-formedness rules listed in Table 5 do not impose any grammar on the two attributes of an action: the pre-condition and the post-condition. To give an idea how they may be formally specified, an example is presented in Appendix A.

Table 5. The well-formedness rules of the SeamCAD modeling language

Rule	Description	Related concepts
R1	There is no cycle in the hierarchy via composition of any hierarchical element.	Hierarchical Element
R2	Any element listed as a child element via composition relation of another element must take this element as the parent.	Hierarchical Element
R3	If an element takes another as its parent element via composition relation, the former must be in the child list of the latter.	Hierarchical Element
R4	Any distributed action that takes a given working object as host object must be either its main distributed action or a descendant of the main distributed action of this working object.	Distributed Action and Working Object
R5	Any localized action that takes a given working object as host object must be either its main localized action or a descendant of the main localized action of this working object.	Localized Action and Working Object

R6	Any property that takes a given working object as host object must be either its main property or a descendant of the main property of this working object.	Property and Working Object
R7	Component distributed actions must take the same host object as the parent distributed action does.	Distributed Action and Working Object
R8	Component localized actions must take same host object as the parent localized action does.	Localized Action and Working Object
R9	Component properties must take the same host object as the parent property does.	Property and Working Object
R10	Two properties connected by an association must take the same working object as host object.	Association, Property and Working Object
R11	Two actions connected by a transition must have the same parent element.	Transition and Action
R12	If a working object and a distributed action are connected by a participation link, the distributed action being mentioned must take the parent working object of the working object being mentioned as host object.	Participation, Distributed Action and Working Object
R13	There is at most one participation link between any pair of a working object and a distributed action.	Participation, Distributed Action and Working Object
R14	There is at most one transition between any pair of two actions.	Transition and Action
R15	The two actions connected by a transition must be of the same kind.	Localized Action, Distributed Action and Transition
R16	A property or a localized action that is related to a distributed action via a goal binding must take a working object that is a component of another working object, which hosts the distributed action being related to by the goal binding, as host object.	Property, Localized Action, Distributed Action and Working Object
R17	A distributed action and a localized action that are related via a means binding must take the same working object as host object.	Localized Action, Distributed Action and Working Object
R18	Two model elements connected by a composition must be of the same kind	Hierarchical Element and Composition
R19	There is always a composition that relates a model element and each of its component elements	Hierarchical Element and Composition

3.3.2. Instantiation and well-formedness in the enterprise model of the online bookstore

This subsection exemplifies the way concrete model elements of the enterprise model of the online bookstore are instantiated from the meta-model of the SeamCAD modeling language and how the well-formedness rules of the meta-model are respected.

Table 6. Concrete model elements of the enterprise model of the online bookstore

Building block		**Concrete model element**
Working Object		BookCoMarket, Bookstore Value Network, Customer, BookCo, ShipCo, PubCo, OpApp, Clerk, SearchServlet, Registration.JSP, Order.JSP, JDBC
Property	Stateless	SellTxn, BuyTxn, MarketTxn, DeliveryTxn, InvoicingTxn, PackagingTxn, InvoiceTxn, HandleTxn, PickTxn
	Stateful	Order, Customer's info, BookCatalog, BookSpec1, BookSpec2, BookSpec3
Distributed Action		sale, mfg_sell, market, run, operate, pick
Localized Action		Sell, Buy, Market, Delivery, Supply, Invoicing, Packaging, Invoice, Pick, Handle
Association		between Order and Customer's info, between Order and BookSpec1
Transition		between Pick and Invoice of PurchasingDep, between Pick and Handle of WarehouseDep,
Participation		between sale and Bookstore Value Network, between sale and Customer, between mfg_sell and BookCo, between mfg_sell and ShipCo, between mfg_sell and PubCo, between market and PurchasingDep, between market and WarehoouseDep, between operate and OpApp, between operate and Clerk
Goal binding		between sale and Sell, between sale and SellTxn, between sale and Buy, between sale and BuyTxn, between mfg_sale and Market, between mfg_sale and MarketTxn, between mfg_sale and Delivery, between mfg_sale and DeliveryTxn, between mfg_sale and Supply, between mfg_sale and SupplyTxn, between market and Invoicing, between market and InvoicingTxn, between market and Packaging, between market and PackagingTxn, between procure and Invoice, between procure and InvoiceTxn, between procure and Handle, between procure and HanldeTxn, between pick and Pick, between pick and PickTxn, etc…
Means binding		between mfg_sell and Sell, between market and Market, between operate and Invoicing, between run and Work

Table 6 lists concrete model elements and relations of the enterprise model of the online bookstore. They are classified by the building blocks from which they are instantiated. By showing that the well-formedness rules of the SeamCAD modeling language are respected, Table 7 illustrates that the enterprise model is well-formed.

Table 7. Well-formedness rules are respected in the enterprise model of the online bookstore

Rule	How the rule is respected?
R1	There is no cycle along the the organizational hierarchy of working objects BookCoMarket, Bookstore Value Network, BookCo, PurchasingDep, OpApp and SearchService. There is no cycle along the functional hierarchy of distributed actions market, procure and pick; of localized actions Invoicing, Pick and Invoice; of localized actions Packaging, Pick and Handle.
R2 R3	Bookstore Value Network and Customer are the component working objects of BookCoMarket. BookCoMarket is their parent working object. BookCo, ShipCo and PubCo are the component working objects of Bookstore Value Network. Bookstore Value Network is their parent working object. PurchasingDep and WarehouseDep are the component working objects of BookCo. BookCo is their parent working object.
R4 R7	BookCo is the host object of distributed actions market, pick and procure.
R5 R8	PurchasingDep is the host object of localized actions Invocing, Pick and Invoice. WarehouseDep is the host object of localized actions Packaging, Pick and Handle.
R6 R9	PurchasingDep is the host object of properties Order, Customer's info, BookCatalog, BookSpec1, InvocingTxn, PickTxn and InvoiceTxn. WarehouseDep is the host object of properties PackagingTxn, PickTxn and HandleTxn.
R10	PurchasingDep is the host object of properties Order, Customer's info and BookSpec1.
R11	Invocing and Pick have the same parent - localizd action Invoice.
R12	sale is hosted by BookCoMarket, which is the parent working object of Bookstore Value Network and Customer. mfg_sell is hosted by Bookstore Value Network, which is the parent working object of BookCo, ShipCo and PubCo. market is hosted by BookCo, which is the parent working object of PurchasingDep and WarehouseDep.
R13	There is at most one participation link between any pair of a working object and a distributed action.
R14	There is at most one transition between any pair of two actions.
R15	Pick and Invoice are localized actions. Pick and Handle are localized actions.
R16	The localized action Sell takes Bookstore Value Network as host object, which is a component working object of BookCoMarket, which hosts sale. The stateless property SellTxn takes Bookstore Value Network as host object, which is a component working object of BookCoMarket, which hosts sale. The localized action Buy takes Customer as host object, which is a component working object of BookCoMarket, which hosts sale. The stateless property BuyTxn takes Customer as host object, which is a component working object of BookCoMarket, which hosts sale.

R17	The distributed action `mfg_sell` and the localized action `Sell`, which are related by a means binding, take `Bookstore Value Network` as host object. The distributed action `market` and the localized action `Market`, which are related by a means binding, take `BookCo` as host object. The distributed action `operate` and the localized action `Invoicing`, which are related by a means binding, take `PurchasingDep` as host object. The distributed action `run` and the localized action `Work`, which are related by a means binding, take `OpApp` as host object.
R18	`BookCo`, `ShipCo`, `PubCo` and `Bookstore Value Network` are all working objects.
R19	Composition between `BookCo` and `Bookstore Value Network`, etc…

3.4. Formalization

In this section, the meta-model of the SeamCAD modeling language is formalized in Alloy. The formalization code that is written in Alloy can be executed.

3.4.1. Formalization in Alloy

Representing the meta-model by means of class diagrams is widely-used techniques for capturing building blocks of the language and possibly some relation between them. Unfortunately, the meta-model presented in the previous section is not enough to capture the well-formedness rules. It is necessary to formalize these rules. Either we take an additional formal language like OCL [26] to declare the rules in combination with the class diagram representing the meta-model, or we can use a declarative language to wholly formalize the language in terms of both the language constructs and the well-formedness rules.

Alloy is a lightweight declarative language based on set and relation theory [9]. It comes with a tool called Alloy Analyzer[7] that can execute Alloy code to either generate an instance model or to find a counter example in a limited domain. Being able to generate an instance model for the Alloy code that formalizes our modeling language implies that the meta-model is consistent at least for the specified domain and thus a concrete model can be made using the modeling language.

A model coded in Alloy typically has two main parts. The first part is the declaration of concepts. In Alloy, keyword `sig`, which stands for Alloy signature, declares a concept having fields that are always considered as sets. All elements of a set of a field are of a specific type, which is defined by another signature or even the same signature where the field being mentioned is declared. The second part is definitions of rules that govern the way any instance model of declared concepts should be. These rules are described as facts (with keyword `fact`) in Alloy. The syntax of Alloy facts is very similar to that of the first order logic. Note that according to the syntax of the Alloy language, these two parts are not necessarily separated. They can syntactically be interleaved to better expose the semantics of the Alloy code that are related to a specific Alloy signature.

[7] Alloy Analyzer at MIT http://alloy.mit.edu

Formalization of Model Elements

The model elements can be formalized in a straightforward way. Each concept expressed in the UML diagram shown in Figure 14 is declared under an Alloy signature. An association end in this diagram is mapped to an Alloy field of the Alloy signature that declares the concept from which the association comes. Any cardinality specified for an association end is mapped to an appropriate Alloy keyword: (* mapped to `set`, 0..1 mapped to `lone`). The generalization in the UML diagram can be mapped to Alloy keyword `extends`.

First, a generic model element is defined. Signature `seamHierarchicalElement` represents a generic model element from which more concrete model elements are derived. A generic model element has component elements (field `components`) and a parent element (field `parent`). Another signature - `seamAction` is declared for the generic action from which the localized action and the distributed action are specialized. This signature is a subtype of `seamHierarchicalElement`.

```
sig seamHierarchicalElement {
   components : set seamHierarchicalElement,
   parent : lone seamHierarchicalElement
}

sig seamAction extends seamHierarchicalElement {
}
```

Next, modeling terms of our modeling language are declared as subtypes of one of the two signatures declared above. Each working object has at most one distributed action, one localized action and one property. Each distributed action, localized action or property refers to a working object in which it is defined. In addition, a property or a localized action refers to at most one distributed action via a goal binding, and a distributed refers to at most one localized action via a means binding.

```
sig seamWorkingObject extends seamHierarchicalElement {
   main_distributed_action : lone seamDistributedAction,
   main_property : lone seamInfoObject,
   main_localized_action : lone seamLocalizedAction
}

sig seamProperty extends seamHierarchicalElement {
   host_object : one seamWorkingObject,
   goal_binding : one seamDistributedAction
}

sig seamDistributedAction extends seamAction {
   host_object : one seamWorkingObject,
   means_binding : one seamLocalizedAction
}

sig seamLocalizedAction extends seamAction {
   host_object : one seamWorkingObject,
   goal_binding : one seamDistributedAction
}
```

All the well-formeness rules that matter over the concepts that are declared by the 5 signatures above will also be formalized in this subsection.

Formalization of Relations

The way relations are formalized should be done in the same way model elements are. First, a generic relation `seamRelation` is declared in the following Alloy line of code.

```
sig seamRelation {}
```

Specific relations are declared in the following code fragment. In principle, each relation has a source element and a destination element, both of which are either working object, property, distributed action or localized action. For the signature that declares a generalization, the union operator in the expression `seamProperty + seamWorkingObject` signifies that the source element and the destination element can either be a property or a working object.

Unlike other relations, the composition can be applied to any kind of model element. For this reason, the source and destination element of a composition are declared as variables of the generic element.

```
sig COMPOSITION extends seamRelation {
   source : one seamHierarchicalElement,
   destination : one seamHierarchicalElement
}

sig ASSOCIATION extends seamRelation {
   source : one seamProperty,
   destination : one seamProperty
}

sig GENERALIZATION extends seamRelation {
   source : one seamProperty + seamWorkingObject,
   destination : one seamProperty + seamWorkingObject
}

sig TRANSITION extends seamRelation {
   source : one seamAction,
   destination : one seamAction
}

sig PARTICIPATION extends seamRelation {
   source : one seamWorkingObject,
   destination : one seamDistributedAction
}
```

Formalization of Well-formedness Rules

The 19 well-formedness rules of the SeamCAD modeling language are formalized using Alloy facts. Syntactically, each Alloy fact has a name and a body, which consists of statements. Semantically, the way these facts are formulated should be the same way as English sentences are transformed into first-order logic statements. Nearly all statements of the Alloy facts that formalize the well-formedness rules have the quantification $\forall$ of the first-order logic (keyword `all` in Alloy).

First, the rule R1 states that the hierarchy made by hierarchical elements must not be cyclic. The `acyclic` fact says that no hierarchical element that can be found in either its transitive closure of the `parent` field or its transitive closure of the `components` field.

```
fact acyclic {
   all e: seamHierarchicalElement | e not in (e.^components + e.^parent)
}
```

Next, the rule R2 and R3 are captured by fact `mutual`. This fact makes sure that if a hierarchical element is a child of another element, the latter must be the parent of the former. Note that the Alloy implication construct (with symbol =>) is exploited in the statement of this Alloy fact.

```
fact mutual {
   all e, epc: seamHierarchicalElement | (epc = e.parent => e in epc.
components) and (epc in e.components => epc.parent = e)
}
```

The rule R4 is captured by the fact `da_in_host`. To state that the main distributed action of a working object always refers to it but does not have any parent, an Alloy fact is defined as follows

```
fact da_in_host {
   all wo: seamWorkingObject, j: seamDistributedAction | j =
wo.main_distributed_action => (j.host_object = wo and no j.parent)
}
```

The fact `da_in_host` is furthered to make sure that any distributed action that refers to a given working object must be a descendant of the main distributed action of that working object.

```
fact da_in_host {
   all wo: seamWorkingObject, j: seamDistributedAction | j =
wo.main_distributed_action => (j.host_object = wo and no j.parent)
   all wo: seamWorkingObject, j : seamDistributedAction -
wo.main_distributed_action | j.host_object = wo => j in
wo.main_distributed_action.^components
}
```

A similar fact is added to formalize the rule R5 that matters on properties.

```
fact pr_in_host {
   all wo: seamWorkingObject, p: seamProperty | p = wo.main_property =>
(p.host_object = wo and no p.parent)
   all wo: seamWorkingObject, p : seamProperty - wo.main_property |
p.host_object = p => p in wo.main_property.^components
}
```

Another similar fact is added to formalize the rule R6 that matters on localized actions.

```
fact la_in_host {
   all wo: seamWorkingObject, l: seamLocalizedAction | l =
wo.main_localized_action => (l.host_object = wo and no l.parent)
   all wo: seamWorkingObject, all l: seamLocalizedAction -
wo.main_localized_action | l.host_object = wo => l in
wo.main_localized_action.^components
}
```

The rules R7, R8 and R9 can be captured together in the fact `same_host`. Note that the expression `seamProperty + seamDistributedAction + seamLocalizedAction` yields a set of all properties, distributed actions and localized actions.

```
fact same_host {
   all e, c: seamProperty + seamDistributedAction + seamLocalizedAction | c in
e.components => c.host_object = e.host_object
}
```

The following fact captures the rules R10, R11, R12. Note that the dot join can be concatenated to reach the `host_object` and `parent` field.

```
fact no_crossing {
   all r: ASSOCIATION | r.source.host_object = r.destination.host_object
   all r: TRANSITION | r.source.parent = r.destination.parent
   all r: PARTICIPATION | r.source.parent = r.destination.host_object
}
```

In the following Alloy fact, the first statement assures that for any two different participation links, either their source elements are different or their destination elements are different. The second statement implies that for any two different transitions, either their source elements are different or their destination elements are different. So, the rules R13 and R14 are formalized by this fact.

```
fact uniqueness {
   all pl1, pl2 : PARTICIPATION | pl1 != pl2 <=> (pl1.source != pl2.source or
pl1.destination != pl2.destination)
   all lt1, lt2 : TRANSITION | lt1 != lt2 <=> (lt1.source != lt2.source or
lt1.destination != lt2.destination)
}
```

To formalize the rule R15, an Alloy fact is added. In the following Alloy code fragment, the code block right after the declaration of signature `TRANSITION` is actually an unnamed fact that is stated in the context of this signature. The two statements in the body of this fact force the source and destination element of a transition to be of the same kind of action.

```
sig TRANSITION extends seamRelation {
   source : one seamAction,
   destination : one seamAction
} {
   source in seamDistributedAction <=> destination in seamDistributedAction
   source in seamLocalizedAction <=> destination in seamLocalizedAction
}
```

The following Alloy fact formalizes the rules R16 and R17. The first statement says that for any localized action `la` and any distributed action `da`, if they are related by a means binding, then they have the same host object. The second statement states that for any property `p` and any distributed action `da`, if they are related by a goal binding, then they have the same host object. The third statement implies that for any localized action `la` and any distributed action `da`, if they are related by a goal binding, then they have the same host object.

```
fact goal_means {
   all la: seamLocalizedAction, da: seamDistributedAction |
            da.means_binding = la => da.host_object = la.host_object
   all p: seamProperty, da: seamDistributedAction |
            p.goal_binding = da => p.host_object.parent = da.host_object
   all la: seamLocalizedAction, da: seamDistributedAction |
            l.goal_binding = da => la.host_object.parent = da.host_object
}
```

The code block right after the declaration of signature `COMPOSITION` is actually an unnamed fact that is stated in the context of this signature. All fields of this signature can be referred to without being prefixed by a variable of `COMPOSITION` and a dot symbol. This fact assures that the source and the destination of a composition must be variable of the same kind of model element, which is actually what the rule R18 means.

```
sig COMPOSITION extends seamRelation {
   source : one seamHierarchicalElement,
   destination : one seamHierarchicalElement
} {
   source in seamWorkingObject <=> destination in seamWorkingObject
   source in seamProperty <=> destination in seamProperty
   source in seamDistributedAction  <=> destination in seamDistributedAction
   source in seamLocalizedAction  <=> destination in seamLocalizedAction }
```

To capture the rule R19 (make sure that there is always a composition between any pair of model elements that have parent-child relationship), a fact called `composition_link` is defined as follows

```
fact cmps_link {
   all p, ch: seamHierarchicalElement | ch.parent = p <=> (some c: COMPOSITION
| c.source = p and c.destination = ch)
}
```

It is important to keep track of how the well-formedness rules that were presented in the meta-model are formally declared in Alloy. Table 8 shows the mapping from these rules to Alloy facts that have been explained in this section.

Table 8. Mapping between informal well-formedness rules of the SeamCAD modeling language to Alloy facts

Rule	Brief semantics	Alloy fact
R1	No cycle in hierarchy	`acyclic`
R2	Parent is the inverse of children	`mutual`
R3	Children is the inverse of parent	`mutual`
R4	Hierarchy of distributed action	`da_in _host`
R5	Hierarchy of localized action	`la_in _host`
R6	Hierarchy of property	`pr_in _host`
R7	Working object of distributed actions	`same_host`
R8	Working object of localized actions	`same_host`
R9	Working object of property	`same_host`
R10	No cross-boundary association	`no_crossing`
R11	No cross-boundary transition	`no_crossing`
R12	No cross-boundary participation link	`no_crossing`
R13	No more than one participation link	`uniqueness`
R14	No more than one transition between any two actions	`uniqueness`
R15	A transition connects actions of the same kind	attached to signature
R16	Goal binding spreading over two working objects	`goal_means`
R17	Means binding for one working object	`goal_means`
R18	A composition connects two model elements of the same kind	attached to signature
R19	One composition for a model element and each of its components	`cmps_link`

Complete Alloy Code

Putting all the declarations, facts and predicate that were presented above results in the whole Alloy code that. Note that the orders in which these declarations and facts are put together are flexible. To check the consistency of the Alloy code that was presented in the two previous subsections, we can define some Alloy predicate to generate some instance model. If the Alloy Analyzer can generate an instance model, even a trivial one, the Alloy code is syntactically correct and not over-constrained at least for the domain specified when executing the code. In the following code fragment, predicate `trivial` instantiates a trivial model.

```
pred trivial() {}

run trivial
```

Figure 15 gives the complete Alloy code that formalizes the SeamCAD modeling language. The code has 12 Alloy signatures, 12 Alloy facts, 28 statements and 130 lines of code.

```
module seamcad_meta

sig seamHierarchicalElement {
   components : set seamHierarchicalElement,
   parent : lone seamHierarchicalElement
}

sig seamAction extends seamHierarchicalElement { }


fact acyclic {
   all e: seamHierarchicalElement | e not in (e.^components + e.^parent)
}

fact mutual {
   all e, epc: seamHierarchicalElement | (epc = e.parent => e in epc.
components) and (epc in e.components => epc.parent = e)
}

sig seamWorkingObject extends seamHierarchicalElement {
   main_distributed_action : lone seamDistributedAction,
   main_property : lone seamInfoObject,
   main_localized_action : lone seamLocalizedAction
}

sig seamProperty extends seamHierarchicalElement {
   host_object : one seamWorkingObject,
   goal_binding : one seamDistributedAction
}

sig seamDistributedAction extends seamAction {
   host_object : one seamWorkingObject,
   means_binding : one seamLocalizedAction
}

sig seamLocalizedAction extends seamAction {
   host_object : one seamWorkingObject,
   goal_binding : one seamDistributedAction
}

fact da_in_host {
   all wo: seamWorkingObject, j: seamDistributedAction |
      j = wo.main_distributed_action => (j.host_object = wo and no j.parent)
}

fact pr_in_host {
   all wo: seamWorkingObject, p: seamProperty | p = wo.main_property =>
          (p.host_object = wo and no p.parent)
   all wo: seamWorkingObject, p : seamProperty - wo.main_property |
          p.host_object = wo => p in wo.main_property.^components
}

fact la_in_host {
   all wo: seamWorkingObject | all l: seamLocalizedAction |
     l = wo.main_localized_action => (l.host_object = wo and no l.parent)
   all wo: seamWorkingObject | all l: seamLocalizedAction -
wo.main_localized_action | l.host_object = wo => l in
wo.main_localized_action.^components
}

fact same_host {
   all e, c: seamProperty + seamDistributedAction + seamLocalizedAction | c in
e.components => c.host_object = e.host_object
}

sig seamRelation {}

sig ASSOCIATION extends seamRelation {
   source : one seamProperty,
   destination : one seamProperty
}

sig GENERALIZATION extends seamRelation {
   source : one seamProperty + seamWorkingObject,
   destination : one seamProperty + seamWorkingObject
}

sig TRANSITION extends seamRelation {
   source : one seamAction,
   destination : one seamAction
} {
   source in seamDistributedAction <=> destination in seamDistributedAction
   source in seamLocalizedAction <=> destination in seamLocalizedAction
}

sig PARTICIPATION extends seamRelation {
   source : one seamWorkingObject,
   destination : one seamDistributedAction
}

fact no_crossing {
   all r: ASSOCIATION | r.source.host_object = r.destination.host_object
   all r: TRANSITION | r.source.parent = r.destination.parent
   all r: PARTICIPATION | r.source.parent = r.destination.host_object
}

fact uniqueness {
   all pl1, pl2 : PARTICIPATION | pl1 != pl2 <=> (pl1.source != pl2.source or
pl1.destination != pl2.destination)
   all lt1, lt2 : TRANSITION | lt1 != lt2 <=> (lt1.source != lt2.source or
lt1.destination != lt2.destination)
}

fact goal_means {
   all la: seamLocalizedAction, da: seamDistributedAction |
              da.means_binding = la => da.host_object = la.host_object
   all p: seamProperty, da: seamDistributedAction |
              p.goal_binding = da => p.host_object.parent = da.host_object
   all la: seamLocalizedAction, da: seamDistributedAction |
              l.goal_binding = da => la.host_object.parent = da.host_object
}

sig COMPOSITION extends seamRelation {
   source : one seamHierarchicalElement,
   destination : one seamHierarchicalElement
} {
   source in seamWorkingObject <=> destination in seamWorkingObject
   source in seamProperty <=> destination in seamProperty
   source in seamDistributedAction  <=> destination in seamDistributedAction
   source in seamLocalizedAction  <=> destination in seamLocalizedAction
}

fact cmps_link {
   all p, ch: seamHierarchicalElement | ch.parent = p <=> (some c: COMPOSITION
| c.source = p and c.destination = ch)
}

pred trivial() {}

run trivial
```

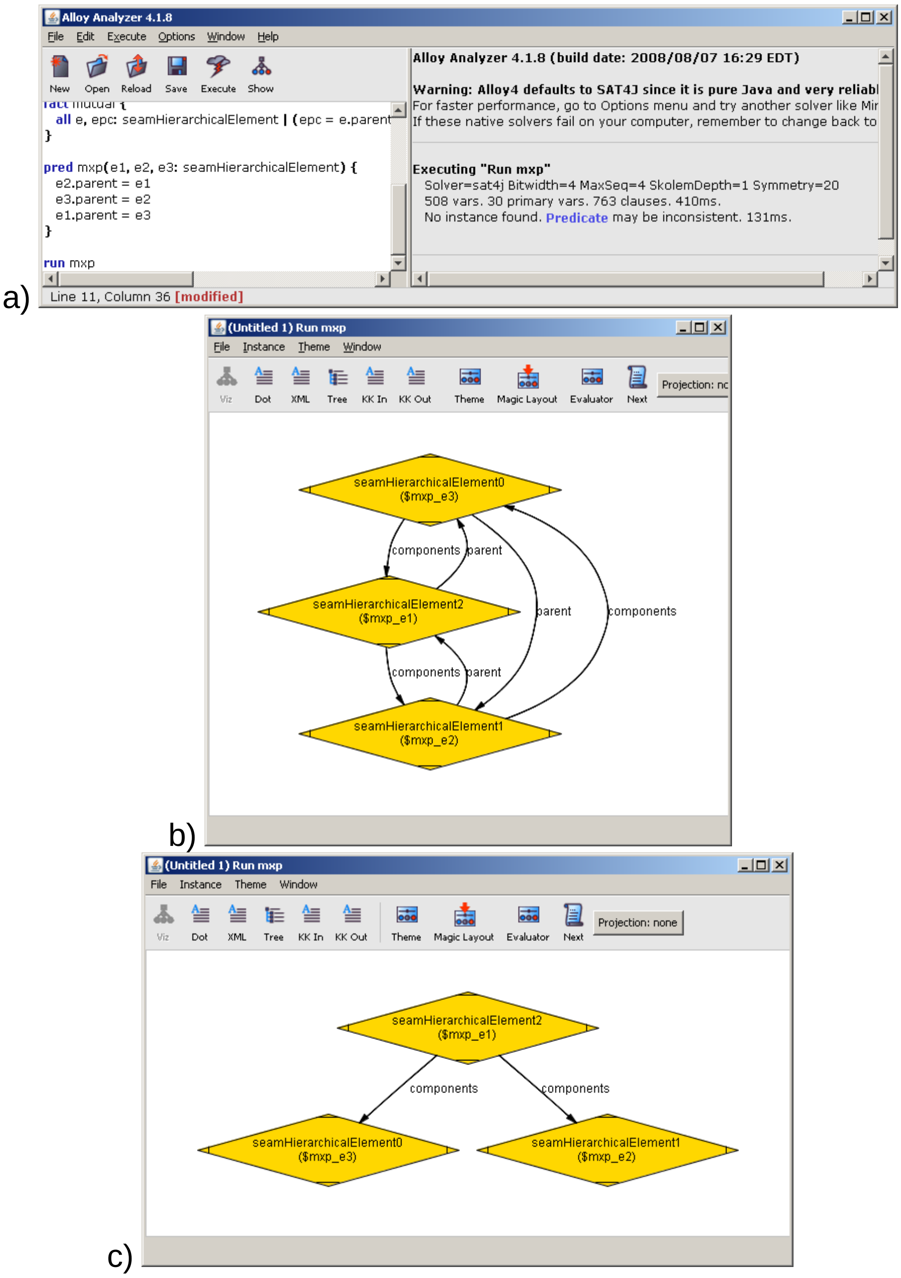
Figure 15. Complete Alloy code formalizing the SeamCAD modeling language

3.4.2. Testing and executing the formalization code

It is possible to test the Alloy code in the other way round by making it overconstrained or underconstrained. This can be done by declaring a predicate specifying an instance model that intentionally violates the declared Alloy facts or by running a weakened version of the Alloy code with the removal of some Alloy facts. Figure 16 a) is the result of running a predicate that specifies an instance model of 3 working objects that have a loop among the parent hierarchy. We can see that Alloy Analyzer cannot find any instance model for this overconstrained code. Figure 16 b) is the result of running an underconstrained version of the Alloy code in which the fact `acyclic` is removed. Figure 16 c) is the result of running another underconstrained version of the Alloy code in which the fact `mutual` is removed. We can see the presence of a cycle and the lack of the link from component working objects to the parent one, respectively.

In addition to testing, we can try to instantiate the Alloy code that formalizes the meta-model of the SeamCAD modeling language. If Alloy Analyzer can generate meaningful instance model out of this Alloy code, we can belive that the meta-model does not have contraditions. In subsection 3.3.2, the instances of the model of the online bookstore are described using the SeamCAD modeling language. These instances can be created using the SeamCAD computer-aided tool[8]. It is possible to intuitively verify that the enterprise model built in the tool matches the instance model generated by Alloy Analyzer in an instance-by-instance manner.

[8] The SeamCAD tool is presented in the next chapter



```
fact mutual {
  all e, epc: seamHierarchicalElement | (epc = e.parent
}

pred mxp(e1, e2, e3: seamHierarchicalElement) {
  e2.parent = e1
  e3.parent = e2
  e1.parent = e3
}

run mxp
```

Line 11, Column 36 [modified]

```
Alloy Analyzer 4.1.8 (build date: 2008/08/07 16:29 EDT)

Warning: Alloy4 defaults to SAT4J since it is pure Java and very reliabl
For faster performance, go to Options menu and try another solver like Mir
If these native solvers fail on your computer, remember to change back to

Executing "Run mxp"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  508 vars. 30 primary vars. 763 clauses. 410ms.
  No instance found. Predicate may be inconsistent. 131ms.
```

Figure 16. Samples generated by Alloy Analyzer after executing overconstrained Alloy code and underconstrained Alloy code

In the following code fragment, predicate `mxp` instantiates the very first organizational level of the bookstore model. We declare the model elements that we want the Alloy Analyzer tool to instantiate as parameters of this Alloy predicate. In the body of this predicate, Alloy statements define how these models element should be put together in the bookstore example.

```
pred mxp(BOOKMARKET, CUSTOMER, BOOKSTOREVN: one seamWorkingObject,
       sale: one seamDistributedAction,
       SellTxn, Book, BookSpec, Message, CustomerInfo: one seamProperty,
       Buy, Sell: one seamLocalizedAction, pl1, pl2: one PARTICIPATION) {
    Buy != Sell
    #(SellTxn + Book + BookSpec + Message + CustomerInfo) = 5
    no BOOKMARKET.parent
    no BOOKMARKET.main_localized_action
    no BOOKMARKET.main_property
    BOOKMARKET.main_distributed_action = sale
    no sale.components
    BOOKMARKET.components = BOOKSTOREVN + CUSTOMER
    no BOOKSTOREVN.components
    BOOKSTOREVN.main_localized_action = Sell
    no Sell.components
    BOOKSTOREVN.main_property = SellTxn
    SellTxn.components = Book + BookSpec + Message + CustomerInfo
    no CUSTOMER.components
    no CUSTOMER.main_distributed_action
    CUSTOMER.main_localized_action = Buy
    no CUSTOMER.main_property
    no Book.components
    no BookSpec.components
    no CustomerInfo.components
    pl1.source = BOOKSTOREVN and pl1.destination = sale
    pl2.source = CUSTOMER and pl2.destination = sale
}

run mxp for 11
```

Running the code in Alloy Analyzer yields an instance model that can be visualized as we can see in Figure 17. All the variables that are declared in the header of the predicate mxp are visualized as pictograms. The text inside these pictograms has two lines. The line above is an internal name given by Alloy Analyzer (not so interesting). The line below indicates the name of the variable (in parentheses). There are also pictograms that represent compositions that are in fact not declared in the predicate mxp. These compositions are generated by the Alloy fact cmps_link. Note that the links connecting all these pictograms stand for the Alloy fields of the Alloy signatures of which they are instantiated. Note that we customized the shape of the pictograms to better illustrate that they are instances of different Alloy signatures.

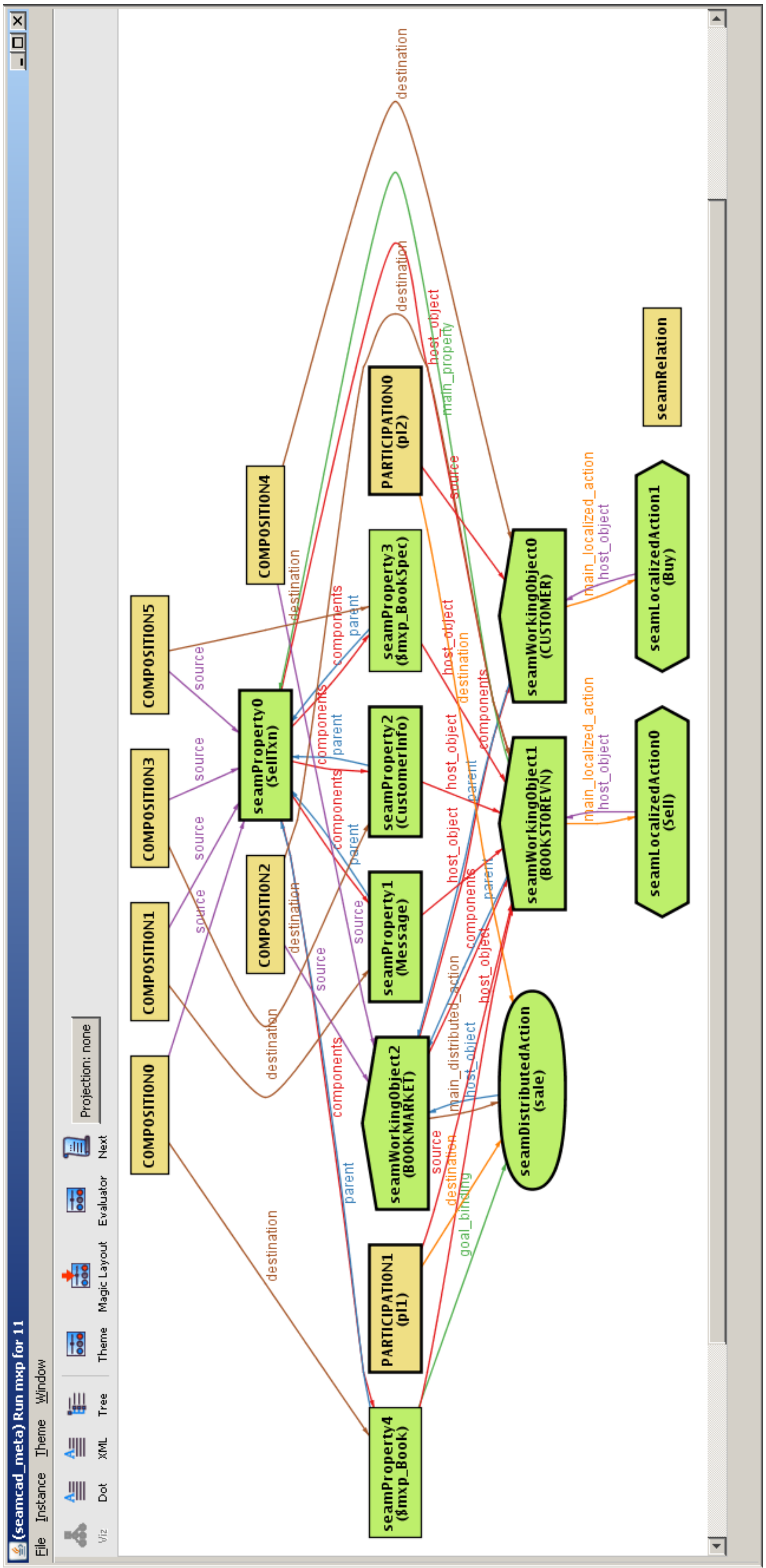


Figure 17. Visualization of the 1st organizational level of the bookstore example generated by Alloy Analyzer

3.5. Notation

The SeamCAD modeling language has a notation scheme, which defines pictograms of different kinds of model element and relation are presented and the rules that mandate the way these pictograms are put together in a diagram.

The notation of the SeamCAD modeling language has two principles: a) to visually express both the organizational hierarchy and the functional hierarchy in diagrams; b) to rely on some widely-used notation in modeling. To meet the first principle, nested pictograms are taken because they visually show the containment. More specifically, to visually show that a model element is a component element of another, the pictogram of the latter surrounds that of the former. There is a popular alternative approach (but less visual) to express the containment. The pictograms of a component model element and its parent element are connected by a line that represents the containment.

To fulfill the second principle, most of pictograms are taken from Unified Modeling Language (UML), which is a widely-practiced modeling language for software and system development. UML has various pictograms for depicting not only systems and their components but only their attributes, properties, states and behaviors. Today, the notations of many modeling languages and development processes take their root from UML.

In the SeamCAD modeling language, the working object may represent any business unit or an IT system, a certain computer application or even a software component. For the working objects that stand for system or software, the UML subsystem notation is the best because it has two meanings: as a classifier and as a package (in the UML meta-model, the subsystem inherits from both the classifier and the package). As a classifier, it represents something that has both structural and behavioral features. As a package, it can group other model elements, including subsystems, like a container. For the working objects that stand for business units such as companies, a block arrow Porter notation is used instead of a UML subsystem notation to make the SeamCAD modeling language closer to business-minded modeler. For the working objects standing for people, the UML actor notation is exploited. Note that it is not interesting to model a human being as whole or as composite like the manner in which a company or a software component is represented. Therefore, the pictogram of people is not nested to show anything inside.

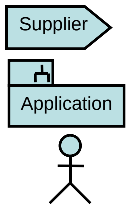
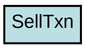
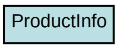
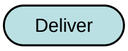
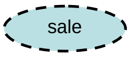
The property in the SeamCAD modeling language takes the UML class notation but the two compartments where the class attributes and the class operations are visible. Specifying association and generalization among properties of the same working object results in a UML class diagram that is rendered inside a pictogram of a working object seen as whole.

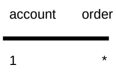
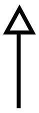
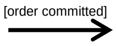
The localized action in the SeamCAD modeling language takes the UML action notation. Specifying transitions among localized actions of the same working object results in a UML activity diagram that is rendered inside a pictogram of a working object seen as whole.

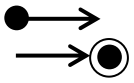
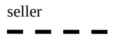
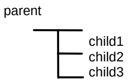
The distributed action in the SeamCAD modeling language takes the UML collaboration notation. Specifying participation links among distributed actions and working objects that either takes a given working object as the host object or are its component objects results in a UML pattern structure that is rendered inside a pictogram of a working object seen as composite.

Table 9 summarizes the pictograms and the syntax of the SeamCAD modeling language notation. The notation syntax is in fact originated from the well-formedness rules of the modeling language, which are captured by facts of the Alloy code that formalizes the SeamCAD modeling language.

Table 9. Building blocks of the SeamCAD modeling language and their notation

Term		Informal Definition	Notation syntax		Notation
			Black-box	White-box	
Working Object		Represents any business unit or IT component found in enterprise systems. They "work" together.	Working Object as whole is characterized by Properties and Localized Actions `R5 & R6 & R8 & R9`	Working Object as composite is characterized by component Working Objects and Distributed Actions `R4 & R7`	
Property	Stateful property	Externally-visible properties that characterize a given Working Object	Property as whole does not show anything nested	Property as composite is characterized by component Properties. They must be of the same working object. `R9`	
	Stateless Property (Transaction)	Representation of the occurrence of a localized action. Transaction is also considered as a context in which normal properties are defined			
Localized Action		Externally-visible actions performed by a given Working Object seen as whole	Localized Action as whole does not show anything nested	Localized Action as composite shows component Localized Actions. They must be of the same working object. `R8`	
Distributed Action		Interaction between multiple working objects that are distributed into localized actions of participating Working Objects seen as composite.	Distributed Action as whole does not show anything nested	Distributed Action as composite shows component Distributed Actions. They must be mediated by the same working object. `R7`	

		Source	Destination	
Association (P-P)	Relation between two properties of the same working object.	A property	A property	
		The two properties must be of the same working object. Rolename and cardinality can be specified for each property. `R10`		
Generalization (P-P) (WO-WO)	Relation between two model elements of the same kind.	A property or a working object	A property or a working object	
Transition (A-A)	Relation between two actions of the same kind that are parented by the same action	A localized action	A localized action	
		The two actions must be parented by the same action. Condition can be specified for the transition. `R11`		

Start / Stop Transition	Relation coming to for going from a localized action	The action must be parented by the action in which the transition is defined. R11		
Participation (WO-A)	Relation between a working object and a distributed action in which it participates.	A working object	A distributed action	
		The working object must be parented by the same working in which the distributed action is defined. Rolename can be specified for the working object. R12		
Composition (WO-WO) (P-P) (A-A)	Relation between a model element and its parent model element.	Working object, property, localized action or a distributed action	The same kind of model element as the source. No loop! R1 & R2 & R3	
		Nested notation is the diagram. Tree-like notation in the model navigation panel.		
Goal binding (P-A) (A-A)	Implicit relation from stateless property (transaction) or a localized action to a distributed action	A stateless property or a localized action	A distributed action	None
		The stateless property represents the occurenece of the localized action, which in turn represents the responsibility of the working object participating in the distributed action. R16		
Means binding (A-A)	Implicit relation from a distributed action to a localized action	A distributed action	A localized action	None
		The distributed action implements the localized action. They should be defined for the same working object. R17		

To illustrate what does the notation syntax means, Figure 18 gives some (but not an exhaustive list) combinations of pictograms that are either legal or illegal in the SeamCAD modeling language. The legal combinations are ticked by a checkmark whereas the illegal ones are marked with a diagonal cross.

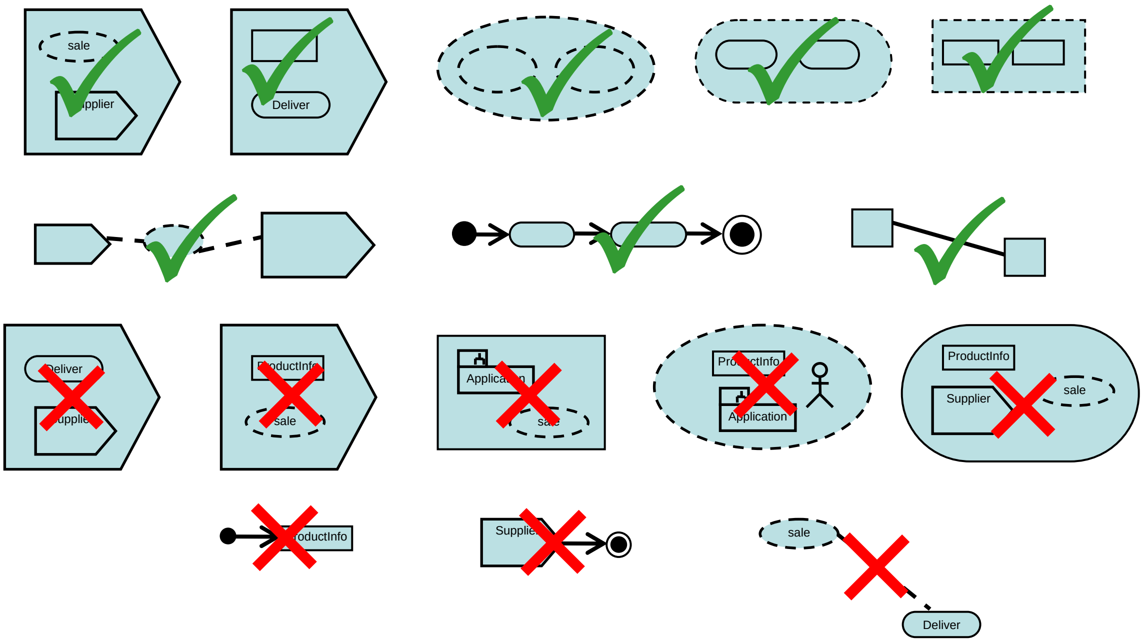


Figure 18. Legal and illegal combinations of pictograms according to the SeamCAD notation

3.6. Solution to Four Modeling Challenges by SeamCAD Modeling Language

In this section, we point out how the SeamCAD modeling language addresses the four modeling challenges presented in Chapter 1. Specifically, the SeamCAD modeling language solves them as follows

- The enterprise model is structured into organizational levels. Each level is represented following the same pattern: working objects that participate in distributed actions (*uniformness*)
- Any working object can be represented either as whole or as composite making it possible to represent multiple business entities and IT systems in the enterprise model (*multi-entity*)
- Any distributed action can be broken down, resulting in detailed representations of localized actions and properties that are bound to it. (*granularity*)
- The well-formedness rules of the language are always enforced to maintain the intrinsic relations in and the well-formedness of the enterprise model (*well-formedness*)

Chapter 4: SeamCAD Computer-Aided Tool

***<u>Overview</u>**: This chapter is dedicated to the SeamCAD tool - a computer-aided tool that was specifically developed to support the SeamCAD modeling language. First, the role of such a computer-aided tool in modeling EA hierarchically is analyzed. This analysis establishes a list of requirements that a computer-aided tool should fulfill. Next, the tool that has been developed for SeamCAD is presented. The way this tool meets the identified requirements is also highlighted. The SeamCAD tool has the following originalities: i) model hierarchy and model overview are made explicit and visible in every window; ii) in a diagram, pictograms of model elements can be nested to explicitly express the hierarchical containment; iii) the tool manages a coherent enterprise model so that it can generate diagrams (but does not keep a list of diagrams) as limited views of the model; iv) multiple diagrams can be opened at the same time and they are kept in synch. A painting algorithm and a layout algorithm that realize these feature are presented. Last, the tool design, its traceability to the Alloy code that formalizes the SeamCAD modeling language and the metrics of the tool implementation and are discussed.*

4.1. The Role of a Computer-Aided Tool in Modeling EA

Let us consider a computer-aided tool that would manage the enterprise models of the online bookstore. Figure 19 describes the tool and the people who would use it, the EA team members. The universe of discourse (UoD) represents the perceived reality of the team members. In the UoD, the team members perceive entities. Examples of entities are markets, value networks, or actions performed by them. These entities are represented as model elements in the enterprise model.

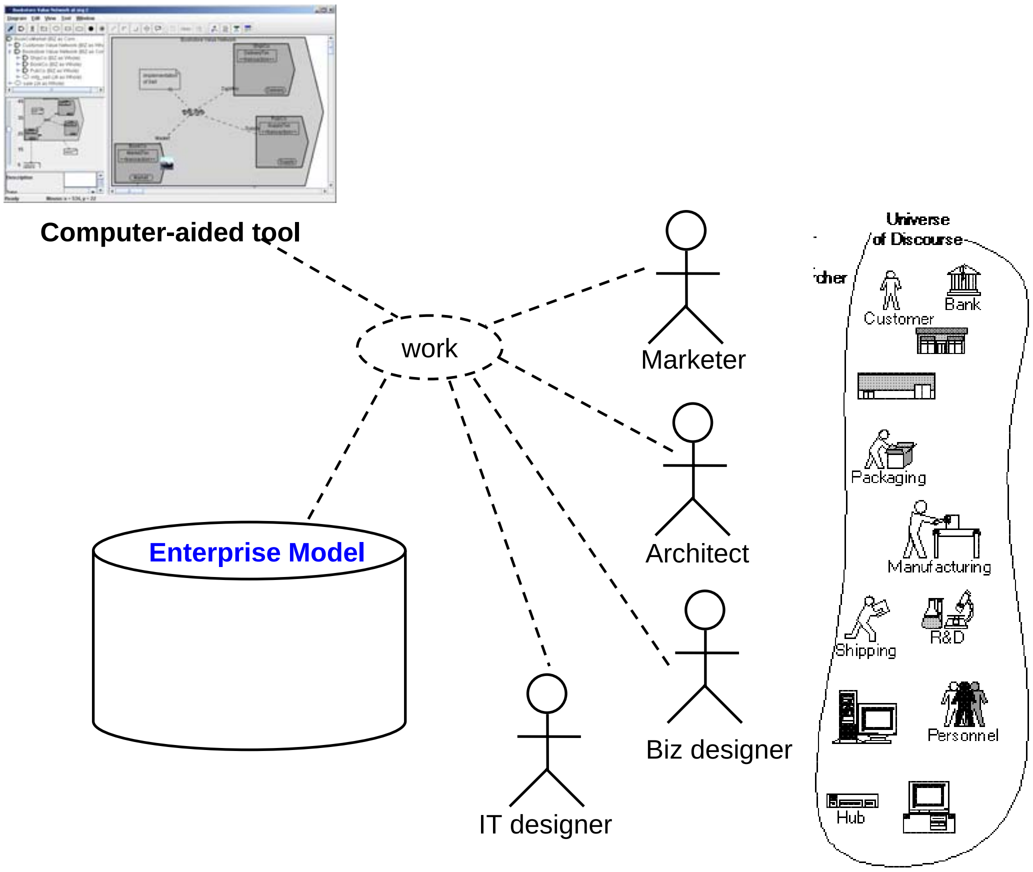


Figure 19. The role that a computer-aided modeling tool plays in an EA multi-discipline team who build a multi-level EA model

In the EA team, there are specialists such as marketers, business process designers, and IT designers. They are responsible for managing specific entities. For instance, the marketers reason about business systems and markets. The business process designers manage business processes. All of them use the tool to build the common enterprise model. In general, each specialist is in charge of a specific level in the model. The enterprise architect coordinates the specialists. Her goal is to insure the alignment between all levels. The tool can help her validate this alignment.

The tool should allow the different specialists to work within the same enterprise model at the level for which they are responsible. It is thus essential that the tool shall explicitly manage an *organizational hierarchy* that represents the enterprise's environment and organization. This is the first requirement.

We have seen in that a system's functionality needs to be modeled at different levels of granularity that make up the *functional hierarchy*. This is the second requirement.

Note that one of the challenges for the tool designer is to provide an ergonomic way to manage these two hierarchies (i.e. enterprise's environment/organization and levels of details in the functionality). If this is not achieved, the modeler might get confused between these two hierarchies.

The members of the EA team expect to reason on graphical representations of the enterprise model. In addition, graphical models are well adapted to represent systems as they make relations between systems more intuitive. This leads to the third requirement that includes the following three characteristics:

- The notation should be systemic meaning that it should be adapted to represent a hierarchical model. For example, all business entities and IT systems could be represented in a uniform way regardless of their nature[9]. The notation should also emphasize concepts such as traceability between levels, relations between a business entity or an IT system and its environment, containment hierarchy…

- The notation should be discipline-specific, so that the specialists can visually recognize what they are responsible of. Although the business entities and IT systems are represented in a uniform manner (e.g. all have properties and participate to actions), the pictograms that represent the different kinds of entity or system can change from one organizational level o another. For example, IT-minded people might want to use UML subsystems to represent IT systems. Business-minded people might want to use the Porter arrow rather than UML subsystems to represent companies.

- The notation should be close to UML whenever possible, so that UML practitioners can have an intuitive feeling of what the notation represents.

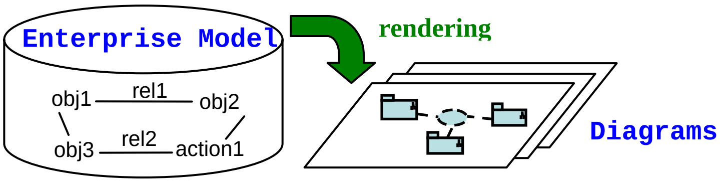


Figure 20. Diagrams are rendered as partial views of the common enterprise model managed by the tool

We also need to specify the way the computer-aided tool manages the model and handles diagrams. In most of the modeling tools, diagrams are normally listed and organized into folders. Quite often, a graphical element such as a class or actor is created in one diagram and will appear in other diagrams. Sometimes, the synchronization between these diagrams creates problems. This synchronization is crucial when modeling hierarchical systems in EA. It is very frequent that elements appear in multiple diagrams. For instance, a company will appear in multiple diagrams. If the name of the company changes, all diagrams need to change. For this reason, diagrams should be generated by extracting the relevant elements and their relationships from a common model (see Figure 20). In addition, the traceability between model elements shown in different diagrams need to be stored in the common, coherent model too. This common model is structured according to the meta-model of the SeamCAD modeling language. This is the fourth requirement.

[9] This is one of the key features of a systemic approach. A systemic approach is based on system theory. The Cambridge Dictionary of Philosophy [27] Audi, R., *The Cambridge Dictionary of Philosophy*: Cambridge University Press, 1999, isbn defines "system theory" as the "trans-disciplinary study of the abstract organization of phenomena, independent of their substance, type, or spatial or temporal scale of existence".

To summarize the requirements, the computer-aided tool shall:

- maintain an explicit organizational level hierarchy that represents the environment and the organization of the enterprise being modeled;
- maintain an explicit functional level hierarchy for the systems that represent the functionality of business entities and IT systems at different levels of granularity;
- implement a notation which is systemic, discipline-specific, understandable by UML practitioners;
- manage a common, coherent model from which the diagrams are generated

4.2. Modeling EA with SeamCAD Tool

This section explains how the SeamCAD tool[10] fulfills the requirements identified in the previous section [28] [29].

4.2.1. Explicit hierarchy that represents the organization and the environment of enterprise

The SeamCAD tool allows the modeler to work on multiple organizational levels. The tool has a main window that shows the organizational level hierarchy of the model in a tree-view widget. This main window enables the user to open editing windows in which some part of the model can diagrammatically be edited.

Figure 21 a) shows an editing window. A tree-view widget to the top left corner of this editing window displays the hierarchy of the model being opened and/or edited. The modeler can interact with it to generate the diagram she wants to display. For example, in Figure 21 a), the working object `Bookstore Value Network` is made context object. The modeler can see in both the tree-view widget and in the diagram that, at the value network level, `Bookstore Value Network` consists of `BookCo` (responsible for purchasing and management), `PubCo` (responsible for providing books) and `ShipCo` (responsible for delivering books) that collaborate together.

[10] The SeamCAD tool is available at http://lamspeople.epfl.ch/lsle/SEAMtool/

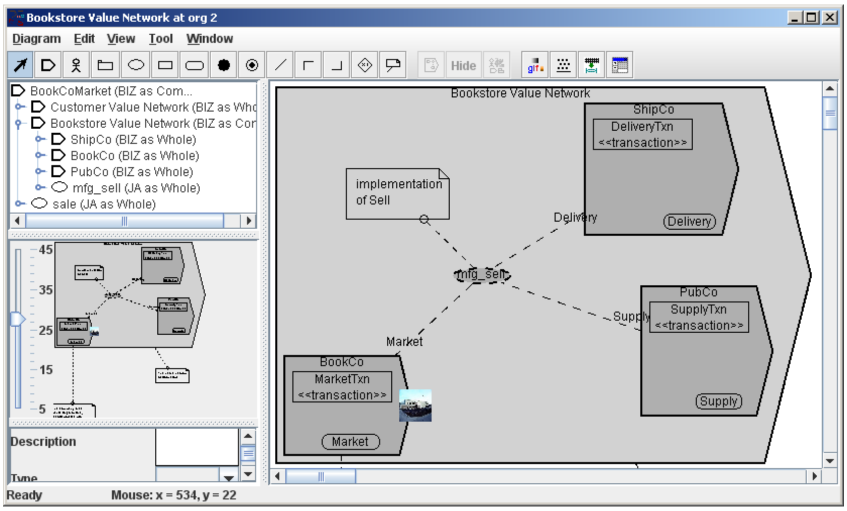
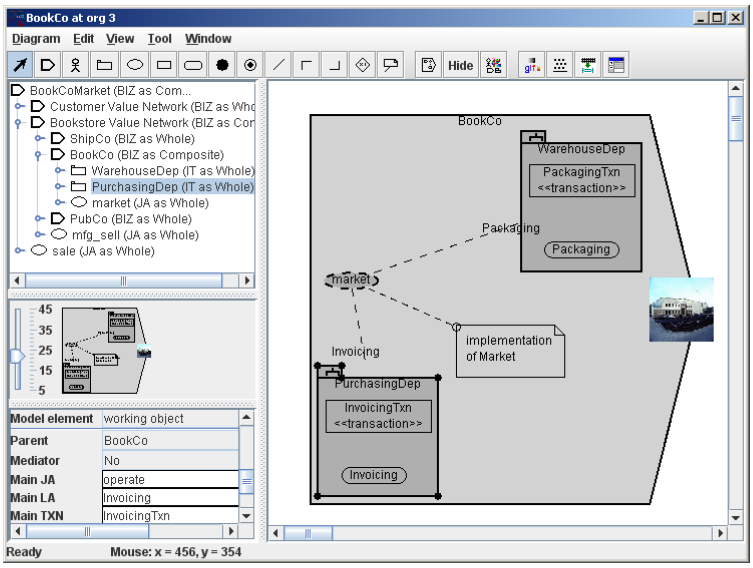


Figure 21 a): `Bookstore Value Network` and its companies seen as wholes



Model element	working object
Parent	BookCo
Mediator	No
Main JA	operate
Main LA	Invoicing
Main TXN	InvoicingTxn

Figure 21 b). `BookCo` and its departments at the company organizational level, functional level 1.

Each editing window is dedicated to a particular organizational level. There is no limit on the number of editing windows opened at the same time. For instance, the modeler can open 3 editing windows and select `Bookstore Value Network`, `BookCo` and `PurchasingDep` as context objects to see the organizational levels described in the example. The tool ensures the consistency among all editing windows. Changes made in any window will propagate to the others.

As can be seen in both Figure 21 a) and Figure 21 b), the top tree node of the tree-view represents the first working object `BookCoMarket` of the organizational hierarchy. The organizational level hierarchy is visible below this node.

The most frequent user interactions in the tree-view are expanding/collapsing a tree node and making a node the context object. If the modeler expands tree node that stands for a working object, it is equivalent to changing to a subsequent organizational level. If the modeler selects a tree node that corresponds to a working object and makes it the context object in the diagram (to the right of the editing window), the environment of this working object is hidden. Note that selections in the tree-view and those in the diagram are always synchronized. For example, the modeler selects the tree node of

BookCo in Figure 21 a), expands it and makes it the context object. The SeamCAD tool then displays a window shown in Figure 21 b) expressing the departmental structure of BookCo - the company level. At this level, there are two departments: PurchasingDep responsible for IT management of customer orders and books, WarehouseDep responsible for inventory processing and packaging. In Figure 21 b), the modeler has purposely chosen to hide the environment of BookCo. In contrast, collapsing a tree node that represents a working object is equivalent to changing to a precedent organizational level.

Figure 21 a) and Figure 21 b) illustrate what we mean by traceability. The localized action Market in BookCo as a whole (in Figure 21 a)) is realized by the distributed action market in BookCo as composite (in Figure 21 b)). In Figure 21, the role of PuchasingDep is Invoicing. This role is also visible as the Invoicing localized action and the InvoicingTxn transaction in PurchasingDep. These relationships are defined as goal bindings and means bindings in the SeamCAD modeling language. In the tool, they are explicitly represented in the data structure of the enterporise model being edited. The modeler only needs to enter once the name of the distributed action (market), the name of the system (PurchasingDep), the name of the localized action (Market) and visually setting the means binding of Market to market. The tool will manage the generation of the note (implementation of Market) and role names (Invoicing).

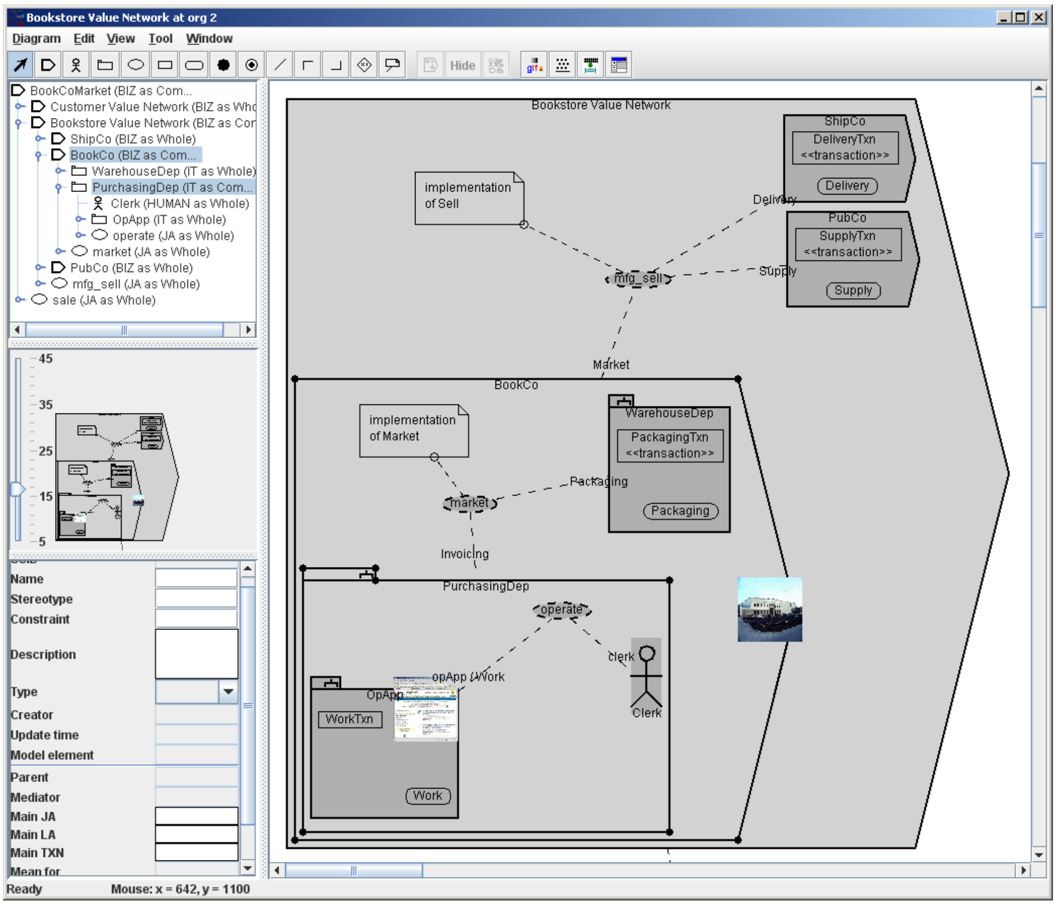


Figure 22. A multi-level representation covering Bookstore Value Network, BookCo, PurchasingDep and OpApp.

It is possible to see multiple organizational levels in one window. Figure 22 shows an editing window in which the value network level, the company level and the department level are represented in one diagram. The user can obtain such a diagram by expanding, in the tree-view widget shown in Figure 21 a), the tree nodes `BookCo` and `PurchasingDep`. This diagram illustrate that the SeamCAD modeling language and tool have a recursive and systematic approach for modeling the organizational hierarchy.

4.2.2. Explicit functional level hierarchy

Navigating through the functional level hierarchy without confusing the modeler is a challenge. Two preliminary versions of SeamCAD were developed until we found adequate solutions to this challenge. These preliminary versions of the SeamCAD implemented organizational level and functional level as completely separate concepts. For each working object, the user could select the functional levels and organizational levels she wanted to display. This lead to problems as it was possible to see diagrams with multiple objects shown at different levels of functionality.

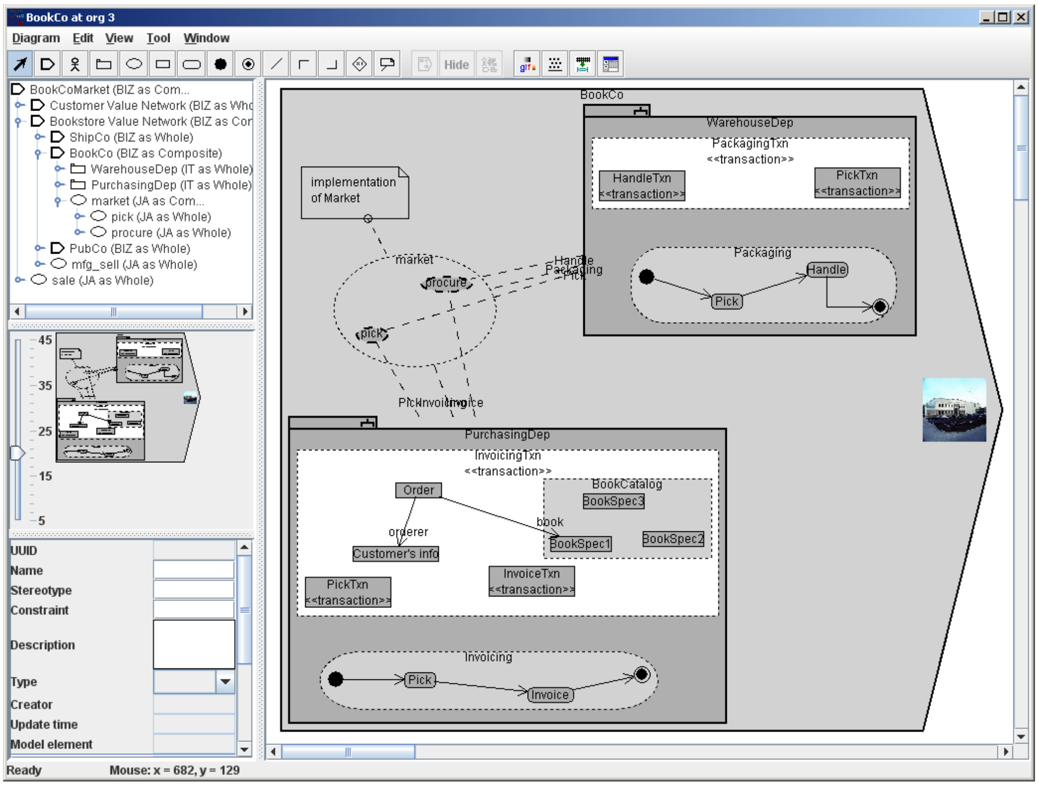


Figure 23. Example of functional level refinement: same organizational level and entities as Figure 21 b) but behaviors described at functional level 2.

We found a solution by enforcing a given level of functionality in all objects shown in the diagram. All working objects participating in this distributed action are displayed at the same level of functionality. This is achieved by giving to the modeler the choice to view the distributed actions as whole or as composite. This feature considerably simplifies the navigation in the functional levels as the concept of functional level is hidden in the notion of distributed action as whole or as composite. It also keeps separate the navigation through functional levels (done by selecting how distributed actions are represented) from the navigation through organizational levels (done by selecting how working objects are represented). The tool relies on the goal bindings that are captured in the model it manages to correctly show all working objects at the same the functional level.

Figure 21 b) and Figure 23 illustrate this point. The `market` distributed action is seen as a composite making the `pick` and `procure` component actions visible. So the information viewpoints of `PurchasingDep` and `WarehouseDep` are seen as composite. Note that these distributed actions have their equivalence in the participating working objects. For example, the `pick` distributed action becomes the `Pick` localized action that represents the service offered by `PurchasingDep`. A stateless property of this department called `PickTxn` represents the occurrences of the corresponding localized action. There is also a property called `BookCatalog` representing the list of books available to the customer from the perspective of `PurchasingDep`. The SeamCAD tool relies on the goal binding going the localized action `Pick` and the stateless property `PickTxn` to the distributed action `pick` to determine that this localized action and this property should be viewed in the same way as the distributed action `pick` is.

4.2.3. A coherent model from which the diagrams are generated

The SeamCAD computer-aided tool manages a coherent model. An overview of the model is visible in every window. We can see that in any of the windows shown Figure 21, Figure 22 and Figure 23, a tree-view widget at the top-left corner shows an overview of the whole enterprise model of the online bookstore. This overview is actually a tree graph of which tree nodes are working objects and distributed actions of the bookstore model. The organizational hierarchy can be seen in this tree graph as a path from, for instance the tree node that stands for `BookCoMarket` to a tree node representing `SerachServlet`. The functional hierarchy can be seen in a similar way. When an editing window is opened, the context working object for this newly-opened window is determined. Then the diagram is rendered starting from the context working object down to the model elements that do not have any component elements.

To make sure that all diagrams are always kept in synch, a propagation mechanism is implemented in the tool. This mechanism is implemented using the Observer/Publisher pattern [30]. Any pictogram is an observer for any change made in the diagram. It will inform the model element in the common model, which play the role of the publisher, about the change. The change is then broadcasted to other pictograms – the observers - in other editing windows to make necessary update. Figure 24 illustrates a situation in which the same working object `BookCo` is painted in two different diagrams. It is seen as whole in the editing window to the left and is seen as composite in the window to the right of Figure 24. Changes made to this working object in one of these windows, for instance renaming it, will automatically propagate to the other window.

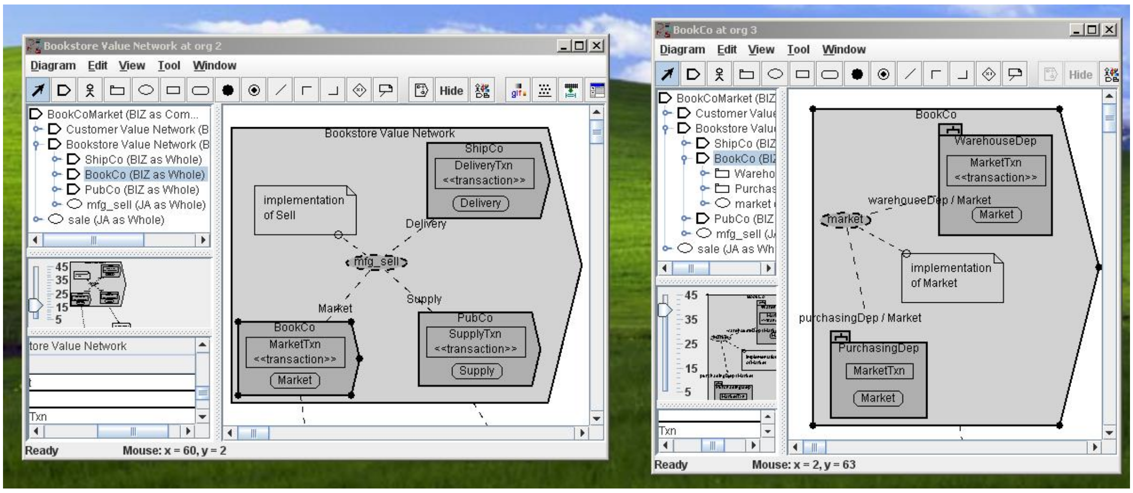


Figure 24. The same working object BookCo appears in two diagrams. Changes made to BookCo in any of these diagrams will automatically propagate to the other diagram.

The modeler has filtering options to control the diagram generation. There are several ways of filtering:

- It is possible to filter out a specific working object or distributed action. Once an element is filtered out, some cognitive change are made in both the tree-view widget and the diagram of the editing window that the filtering is done. In the tree-view widget, the corresponding tree node of the object or action is grayed. In the diagram, the pictogram of the element is hidden. In addition, the pictogram of the parent element becomes transparent. For instance, Figure 25 a) depicts an editing window in which the WarehouseDep is filtered out. We can see that the pictogram of this working object disappears in the diagram to the right and its tree node is grayed in the tree-view widget to the top-left of the editing window.

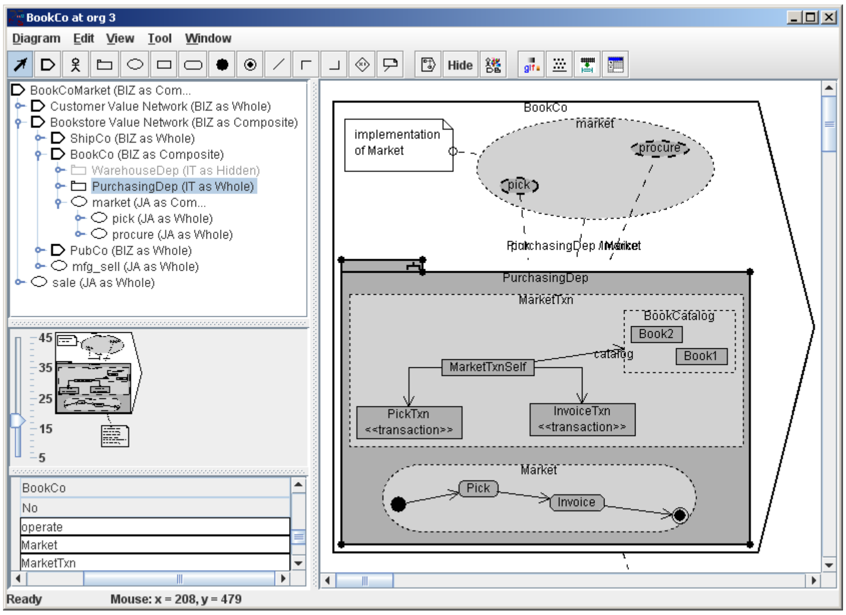


Figure 25 a). Example of information hiding: `WarehouseDep` is hidden.

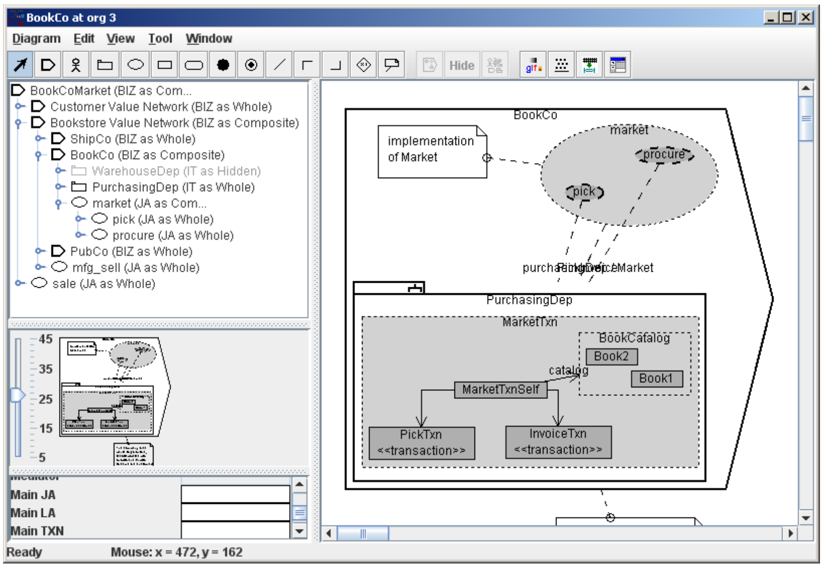


Figure 25 b). `WarehouseDep` is hidden.
The behavior of `PurchasingDep` is hidden too.

- The modeler can decide to hide the environment of a specific working object. This is done when the editing window is created. For example, Figure 21 a) makes the environment of `BookCo` visible whereas Figure 21 b) hides it. Making the environment visible is a powerful feature as it allows the modeler to make the knowledge of the system about its environment explicit. The SeamCAD tool enables drawing a “trace” dependency between an information object in a working object and a model element in the working object’s environment to depict the relationship between a system and its environment.

- A last feature allows the modeler to filter out the property objects or localized actions, or both, of a specific working object. Through this feature, the modeler can obtain diagrams that are close to UML diagrams. Figure 25 b) shows how this kind of filtering is applied to the `PurchasingDep` in the editing window depicted in Figure 25 a). A UML class diagram inside the `PurchasingDep` working object is obtained by hiding its actions. This capability illustrates that the SeamCAD tool could be considered as a UML-like tool in which the context can be systematically represented.

4.2.4. Notation which is systemic, discipline-specific, understandable by UML practitioners

As presented in Section, SeamCAD uses discipline-specific graphical elements to represent the working objects. For instance, in Figure 23, a Porter arrow represents the company and the UML subsystem represents the departments. To make the notation even more concrete, the modeler can attach pictures to a working object. For example, in Figure 22, the plant picture is attached to the pictogram of `BookCo`. This feature helps the modeler to recognize what she is looking at.

The systemic notation has the following features:

- Explicit context representation: model elements such as localized actions, properties and distributed actions are always represented within a working object. The working object makes explicit the system in which these elements are defined. In a similar way, component actions are always represented within the composite action that contains them. This makes the behavioral context in which actions are explicitly defined. For instance, in Figure 23, the `Invoice` and `Pick` localized actions are within `Invoicing` making it visible that they define what `Invoicing` means. This feature is actually realized by the nested notation of SeamCAD. The SeamCAD tool automatically resizes an enclosing pictogram whenever the modeler moves nested ones.
- Representation of multiple entities and systems: the modeler can look simultaneously at the specification of multiple business entities or IT systems at the same time. For instance, it is possible to analyze the behavior of both `PurchasingDep` and `WarehouseDep` in Figure 23.
- Holistic representation of state and behavior: In a working object seen as whole, properties and localized actions can both be visible.

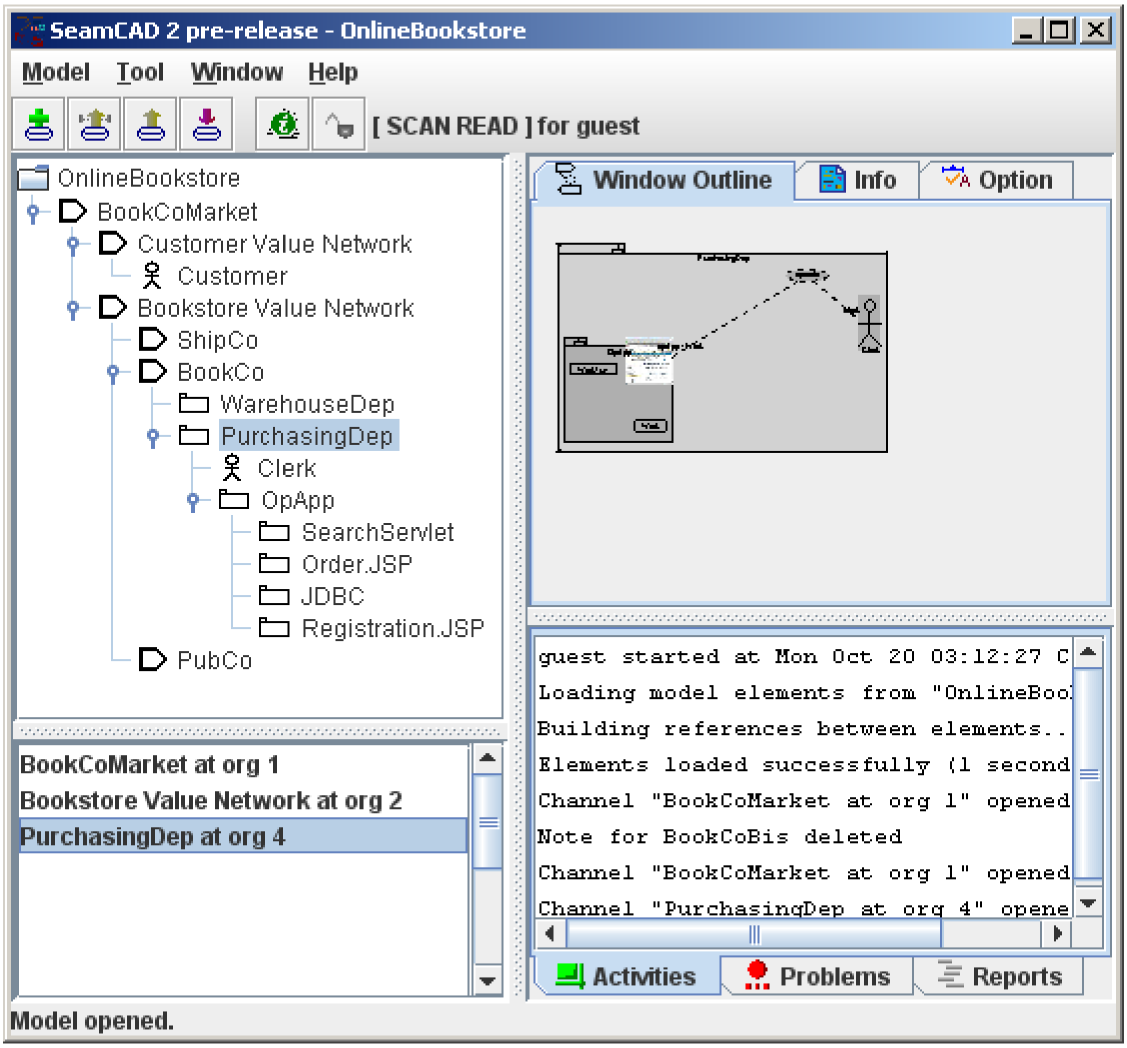
The SeamCAD notation is strongly inspired by UML. Many graphical elements come from UML. The main differences are that the SeamCAD notation permits putting all kinds of model element in a diagram and that SeamCAD pictograms are designed to be nested to visually show the containment hierarchy.

4.2.5. Overview of the model, the diagram and the element

The visibility of the modeler is generally limited and she typically intends to get focused on certain part, not the entire, of the model. It is thus necessary to provide her with some overview of the scope. This principle is relevant at three levels: the model-wide scope, the diagram-wide scope and the element-wide scope.

Overview of Model

Figure 26 is a screenshot of the main window of the SeamCAD tool. The panel in the top left corner is a tree-view widget that gives an overview of the model being opened in the tool. Below it, a list-view widget shows all editing windows being opened. Selecting an item in this list-view will render a small overview in a certain scale of the diagram of the corresponding editing window in a panel to the right of the main window. The modeler can quickly switch to an editing window by invoking a special command (clicking a special button or double-clicking the list item that represents it).



- OnlineBookstore
  - BookCoMarket
    - Customer Value Network
      - Customer
    - Bookstore Value Network
      - ShipCo
      - BookCo
        - WarehouseDep
        - PurchasingDep
          - Clerk
          - OpApp
            - SearchServlet
            - Order.JSP
            - JDBC
            - Registration.JSP
      - PubCo

BookCoMarket at org 1
Bookstore Value Network at org 2
PurchasingDep at org 4

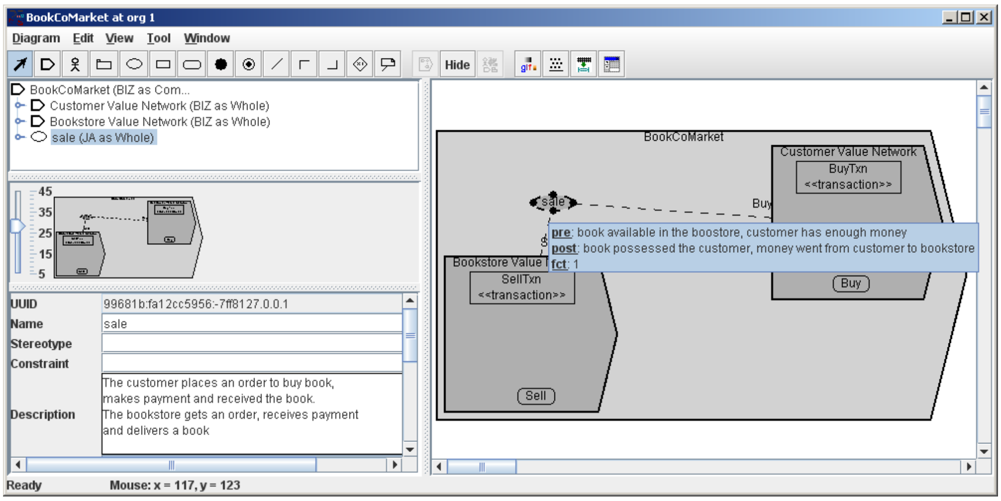
```
guest started at Mon Oct 20 03:12:27 C
Loading model elements from "OnlineBoo
Building references between elements..
Elements loaded successfully (1 second
Channel "BookCoMarket at org 1" opened
Note for BookCoBis deleted
Channel "BookCoMarket at org 1" opened
Channel "PurchasingDep at org 4" opene
```

Model opened.

Figure 26 a). The main window of the SeamCAD tool shows an overview of the model and a list of editing windows being opened.

Overview of Diagram

To the right of each editing window, a graphics panel renders the diagram through which the modeler edits or views her enterprise model. There is another, smaller graphics panel to the left of the editing window (just below the tree-view widget). This panel also renders the same diagram as the panel to the right of the window does but in a smaller scale. The modeler can adjust the rendering scale using a vertical slider to make sure that she can see an overview of the whole diagram in the smaller panel even if the diagram cannot be entirely visible in the bigger one. Note that the diagram rendered in the smaller panel is not interactive in the sense that the modeler can neither move nor edit pictograms of the diagram while she can do so in the bigger graphics panel.



BookCoMarket (BIZ as Com...
Customer Value Network (BIZ as Whole)
Bookstore Value Network (BIZ as Whole)
sale (JA as Whole)

UUID: 99681b:fa12cc5956:-7ff8127.0.0.1
Name: sale
Stereotype:
Constraint:
Description: The customer places an order to buy book, makes payment and received the book. The bookstore gets an order, receives payment and delivers a book

pre: book available in the boostore, customer has enough money
post: book possessed the customer, money went from customer to bookstore
fct: 1

Figure 26 b). An editing window of the SeamCAD tool shows a diagram (to the right). In addition, it shows an overview of the diagram and an overview of the solely selected model element in the diagram.

Overview of Model Element

In the diagrams generated by the SeamCAD tool, each model element is rendered under a pictogram. As the SeamCAD tool specifically implements the SeamCAD modeling language, all the attributes specified for the concepts that are described in the meta-model of the language should be, explicitly or implicitly, rendered by the tool. Basic attributes of the model element including the name and possibly the stereotype are printed inside the pictogram that renders the element. Other attributes such as the pre-condition and the post-condition of an action can be rendered on a tooltip as illustrated in Figure 26 b). When the modeler hovers over the pictogram of the distributed action `sale` (by moving the mouse cursor over the pictogram and stopping for a few seconds), a tooltip is popped up to display the pre-condition and the post-condition (and also the functional level at which the distributed action `sale` is represented) of this action. Note that the SeamCAD tool does not enforce any grammar on the text entered in the pre-condition or post-condition. It does not offer any function to simulate an action neither. Appendix A exemplifies how the pre-condition and the post-condition of an action can be formally specified in terms of properties created in the working object where the action is defined. To view and to change attributes of a model element, the modeler can select its pictogram in the diagram and interacts with a widget panel (at the bottom-left corner of the editing window) that provides her with editable text fields each of which corresponds to an attribute of the model element. There are also non-editable text fields that show the values of the association ends going from the concept of which the selected model element is an instance (e.g. the name of the main distributed action of a working object).

4.2.6. Solution to the four challenges by the SeamCAD tool

In this subsection, we point out how the SeamCAD tool addresses the four modeling challenges that are presented in Chapter 1 and we summarize the originality of the tool. Specifically, the four challenges are solved as follows

- The organizational hierarchy is explicitly shown in an interactive tree-view widget. The modeler can choose any organizational level from this tree-view to work with (*uniformness*)
- The tool allows any working object to be toggled between whole and composite and to become the context object in a diagram. As the working object stands for a business entity, an IT system or a software component, it is possible to design multiple business entities, IT systems ands software components (*multi-entity*)
- The functional hierarchy is explicitly shown in an interactive tree-view widget. The modeler can choose any functional level from this tree-view to work with. In the diagram, properties and localized actions are showed according the distributed action they are bound to. (*granularity*)
- The tool can generate diagrams to partially represent the enterprise model which is kept well-formed by preventing the modeler from doing anything that is against the well-formedness rules defined in the SeamCAD language. The tool keeps diagrams in synch by propagating updates to all diagrams whenever some change is made to the enterprise model in one of the diagrams (*well-formedness*)

The following originalities of the SeamCAD tool enable the solution to the four modeling challenges (uniformness, multi-entity, granularity and well-formedness)

- Model hierarchy and model overview are made explicit and visible in every window
- In a diagram, pictograms of model elements can be nested to explicitly express the hierarchical containment.
- The tool manages a coherent enterprise model so that it can generate diagrams (but does not keep a list of diagrams) as limited views of the model.
- Multiple diagrams can be opened at the same time and they are kept in synch.

To obtain these originalities, the tool needs to be designed with the following consideration.

- Model elements are painted recursively, starting from the context working object. If a model element is out of the scope defined by the context working object, it is not painted.
- The diagram typically has a lot of nested pictograms. Some of them can be nested to more than 2 levels. The tool should have a feature to assist the modeler in getting a good diagram layout.
- The well-formnedness of the model edited in the SeamCAD tool is guaranteed by maintaining the references between model elements whenever a creation, deletion or modification is made to the model.

The first two remarks are addressed in the next section. The third one is discussed in the section that follows the next section.

4.3. Rendering and Layout

This section presents how diagrams in the SeamCAD tool are rendered and how the diagram layout works in the tool.

4.3.1. Diagram rendering

```
operation render_diagram
input cxt_obj // a context working object
begin
   paint (cxt_obj)
end

operation working_object_painting
input wo // a working object
begin
   if wo is seen as whole
   begin
      property_painting(the main property of wo)
      localized_action_painting(the main localized action of wo)
   end
   else
   begin
      distributed_action_painting(the distributed action of wo)
      for all component working objects c of wo
         working_object_painting(c)
   end
   draw graphics for wo
end

operation property_painting
input p // a property
begin
   if p is seen as composite
      for all component working objects c of p
         property_painting(c)
   draw graphics for p
end

operation distributed_action_painting
input da // a distributed action
begin
   if da is seen as composite
      forall component working objects c of da
         property_painting(c)
   draw graphics for da
end

operation localized_action_painting
input la // a localized action
begin
   if la is seen as composite
      forall component working objects c of la
         property_painting(c)
   draw graphics for la
end
```

Figure 27. Pseudo-code of the rendering algorithm

The SeamCAD tool manages a common model that is outlined in the tree-view widget of all editing windows and of the main window. When an editing window is opened, the context working object for this newly-opened window is determined. In most cases, it is the working object that was selected in the window where the open command is issued. Upon opening the editing window, the rendering algorithm works to paint the diagram to the right of the editing window. This is basically a recursive algorithm. Starting from the context working object, component model elements are painted until the properties, distributed actions and localized actions seen as whole are reached. For the working object viewed as whole, its main property and main localized action is painted. For the working object viewed as composite, its main distributed action and its component working objects are painted. For the any model element that is a property, a distributed action or a localized action seen as whole, the recursive branch of the rendering algorithm paints the very this element and then stop. For the any model element that is a property, a distributed action or a localized action seen as composite, the recursive branch of the rendering algorithm paints the very this element and its component elements. Note that the context working object is viewed as composite by default. Attributes of model element that are necessary for painting (e.g. name, stereotype) is fetched from the common model. The rendering algorithm also relies on the references between a model element and its component elements maintained in the data structure of the common model. Thanks to this rendering algorithm, all diagrams are consistently rendered. Figure 27 gives pseudo code of this rendering algorithm.

4.3.2. Automatic layout

```
algorithm Main Loop for Horizontal Scan
   horizontalScan(list l)
   sort(l); //sort list according to x-centers
   element e0 := l(i - 1);
   element e1 := l(i);
   for i := 0 to l.length - 1 do
      if (e0.xCenter < e1.xCenter) then
         overlap := calculateBiggestOverlap(e0, l);
         //calculated according to equation (7)
         for j := i + j to l.length - 1 do
            shift(l(j), overlap)
            //shift all elements starting from j about the overlap to the
right
         end for
      else if (e0.xCenter = e1.xCenter) then
         //don't shift the elements
      else if (e0.xCenter = e1.xCenter) && (e0.yCenter = e1.yCenter) then
         //center-points are the exact same
         overlap := calculateBiggestOverlap(e0, l);
         //calculated according to equation (7)
      end if
   end for

algorithm Main Procedure for Optimized FSA
   layout(list l )
   maxLevel := getLevels(l);
   for i := maxLevel to i = 0 do
      list of lists ll;
      //all lists on this level
      ll := getListsOnLevel(i);
      for j := 0 to ll.lenght - 1 do
         horizontalScan(ll(j));
         verticalScan(ll(j));
         i--;
      end for
   end for
```

Figure 28. Force-Scan algorithm for shifting overlapped pictograms

The modeler typically faces while interacting with diagrams of SeamCAD. Most frequently, the modeler toggles between the two whole and the composite of some model element in the diagram. As the numbers of pictograms nested inside these two views as well as their relative positions to the enclosing pictogram are basically different, the dimension of the pictogram of the model element being toggled computed by the tool may suddenly be changed. This change could result in two extreme cases: a) the new dimension of the pictogram of the model element being toggled is significantly smaller than that before the toggling operation. As a result, the diagram has a lot of unused space between pictograms; b) the new dimension of the pictogram of the model element being toggled is significantly bigger than that before the toggling operation, potentially get overlapped with other pictograms. In these two cases, the diagram layout needs to be optimized.

A master project was carried out to find an automatic layout algorithm and to realize it in the SeamCAD tool [31]. Initially, an algorithm called Force-Scan [32] is defined and implemented to solve the problem of overlapping pictograms in the second extreme case aforementioned. The following gives a brief informal description of the algorithm:

1. Sort all the pictograms according to the values of their x-centers
2. Start the horizontal scan from the left-most node to the right
3. Sort the nodes according to the values of their y-centers
4. Start the vertical scan from the up-most node down to the bottom

The pseudo code of this algorithm can be found in Figure 28. The algorithm is then extended to cope with the first aforementioned extreme case. The resulting algorithm can compute a compact overlapping-free layout. This algorithm is implemented in SeamCAD as a passive command which can be invoked by the modeler whenever she feels an automatic layout is needed. Figure 29 illustrates the effect of this automatic layout feature. Figure 29 a) is a screenshot of an editing window of which diagram shows `Bookstore Value Network` and `Customer Value Network` within `BookCoMarket`. The modeler toggles `Bookstore Value Network`, and gets the diagram shown in Figure 29 b) where the pictogram of the distributed action `sale` is overlapped with that of `Bookstore Value Network`. Note that there is unnecessarily large space between the pictograms of the action `sale` and of the working object `Customer Value Network`. After having invoked the automatic layout feature, the modeler gets a compact layout as shown in Figure 29 c).

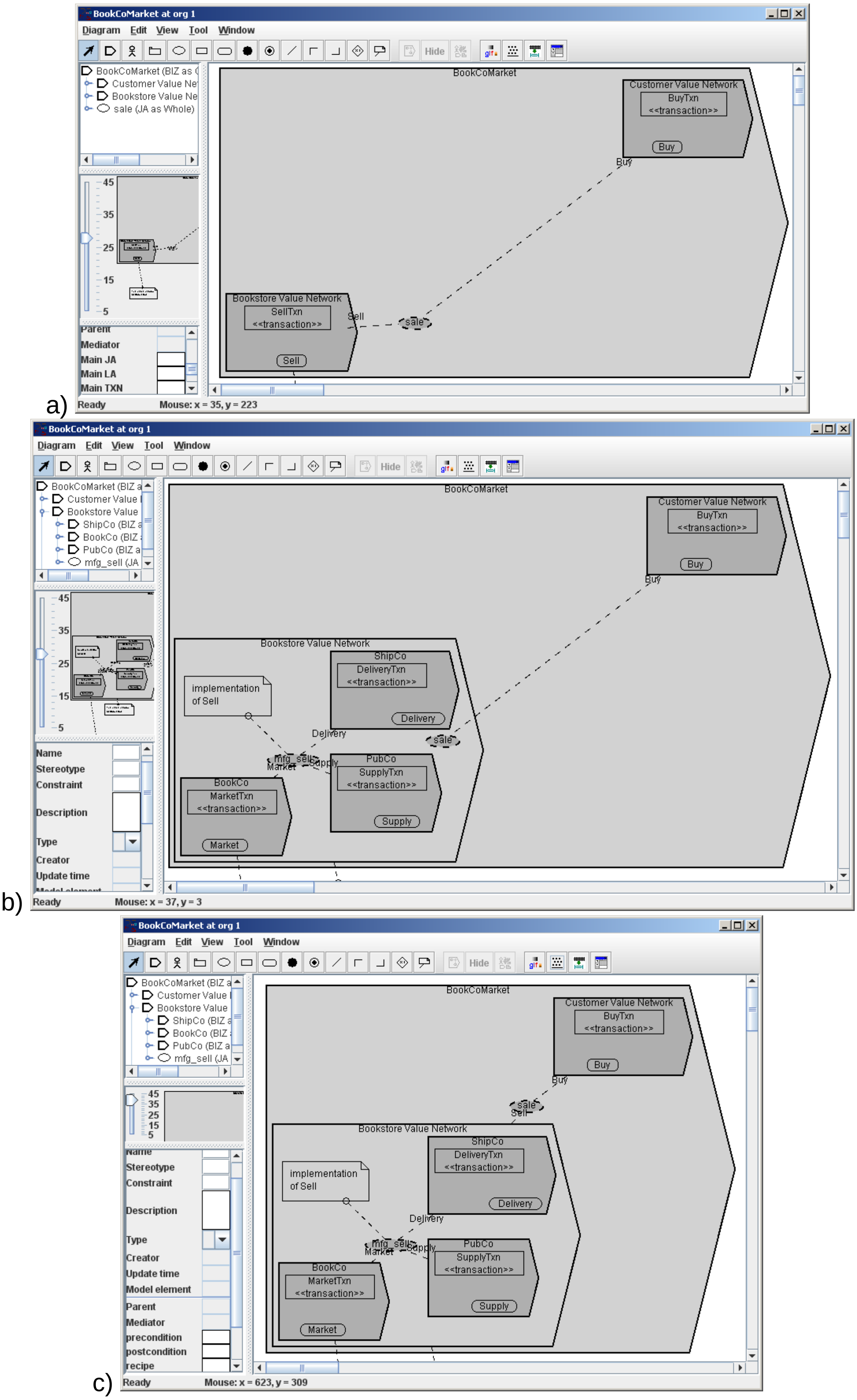


Figure 29. The modeler invokes a command in the tool to change the diagram layout.

The current implementation has some limitation, however. First, if the diagram has many pictograms, the automatic layout feature needs to be invoked multiple times to get the optimal layout. Second, pictograms of the notes are not counted in the layout algorithm.

4.4. Implementation of SeamCAD Tool

We consider the Alloy code that formalizes the meta-model of the SeamCAD modeling language as the formalization or the declarative specification of the SeamCAD modeling framework. On the other hand, as the SeamCAD tool was fully implemented in Java, this Java code can be considered as the realization or the imperative implementation of the SeamCAD modeling framework.

In this section, we present the metrics of the implementation of the SeamCAD tool and compare it to the formalization of the SeamCAD modeling language. A more detailed presentation of the architecture and design of the SeamCAD tool can be found in Appendix D. As presented in Chapter 3, the SeamCAD meta-model consists of a list of building blocks and a list of well-formedness rules. We will discuss the correspondence between the formalization and the realization of these two lists separately.

Table 10. Correspondence between the formalization and the realization of the SeamCAD building blocks

Building block	Brief description	Alloy code	Implementation in Java
Working Object	Represents any business unit or IT system or software component of the enterprise being modeled.	signature `seamWorkingObject`	Java class
Property	Externally-observable properties that characterize a given Working Object.	signature `seamProperty`	Java class
Distributed Action	Represents collaboration between working objects.	signature `seamDistributedAction`	Java class
Localized Action	Externally- observable actions performed by a given Working Object seen as whole.	signature `seamLocalizedAction`	Java class
Expressive relation	Relation that can diagrammatically be expressed including association, participation, transition and generalization	signature `ASSOCIATION`, `PARTICIPATION`, `TRANSITION` and `GENERIALIZATION`	Java class
Intrinsic relation	Goal binding and means binding	fields in signature `seamDistributedAction` `seamLocalizedAction` and `seamProperty`	Java fields

Table 10 shows the correspondence between the formalization and the implementation of the SeamCAD building blocks. All the building blocks but the goal binding and the means binding are formalized by Alloy signatures. Nevertheless, the goal binding and the means binding are formalized by Alloy fields in the signatures formalizing the property, the localized action and the distributed action. In the implementation of the SeamCAD tool, these building blocks are represented in a similar way: all the building blocks but the goal binding and the means binding are realized by Java classes whereas the goal binding and the means binding are realized by Java fields. In other words, there is a straightforward correspondence between the formalization code (in Alloy) and the implementation code (in Java) of the SeamCAD building blocks.

Table 11 shows the correspondence between the formalization and the implementation of the SeamCAD well-formedness rules. In the language specification, all the well-formedness rules are formalized by Alloy facts. In the implementation of the SeamCAD tool, these rules are represented in different ways. For instance, the rule R1 is implemented by the rendering and hit-testing algorithm of the box-in-box notation that prevents the modeler from creating a loop along the organizational and functional hierarchy. The rules R2 and R3 are implemented by some Java code that maintains the integrity of the data structure whenever a creation or a deletion of a model eminent is made. A special Java class offers several methods which can be invoked to check whether a model element can be created inside the pictogram of an existing model element. The checking result is determined based on the specific kind (i.e. working object, property, localized action or distributed action) of the existing model element and the would-be created model element. This class, called the "policy", implements the rules R4, R5 and R6.

Table 11. Correspondence between the formalization and the realization of the SeamCAD well-formedness rules

Rule	Brief semantics	Alloy code	Implementation in Java
R1	No cycle in hierarchy	fact `acyclic`	box-in-box notation rendering and hit-testing algorithm
R2	Parent is the inverse of children	fact `mutual`	reference to parent element is maintained when a new child is added
R3	Children is the inverse of parent	fact `mutual`	reference to parent element is maintained when a new child is added
R4	Hierarchy of distributed action	fact `da_in_host`	Checked by a "policy" class whenever a distributed action is created
R5	Hierarchy of localized action	fact `la_in_host`	Checked by a "policy" class whenever a localized action is created
R6	Hierarchy of property	fact `pr_in_host`	Checked by a "policy" class whenever a property is created
R7	Working object of distributed actions	fact `same_host`	Making host object is a query attribute frees the burden of setting value correct as if it is a field
R8	Working object of localized actions	fact `same_host`	Making host object is a query attribute frees the burden of setting value correct as if it is a field

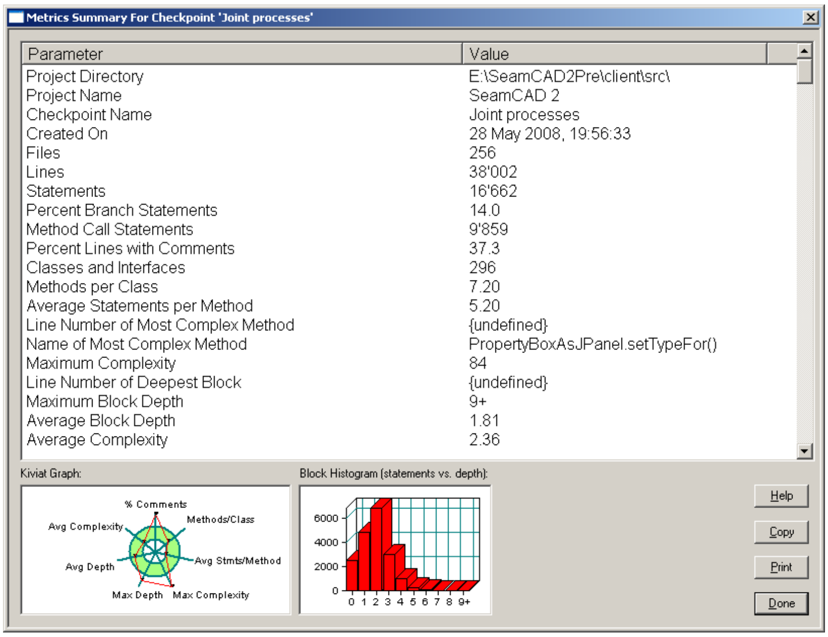
R9	Working object of properties	fact `same_host`	Making host object is a query attribute frees the burden of setting value correct as if it is a field
R10	No cross-boundary association	fact `no_crossing`	
R11	No cross-boundary transition	fact `no_crossing`	
R12	No cross-boundary collaboration link	fact `no_crossing`	
R13	No more than one collaboration link	fact `uniqueness`	Checked by a “policy” class whenever a participation link is made
R14	No more than one transition between any two actions	fact `uniqueness`	Checked by a “policy” class whenever a transition is made
R15	A transition connects actions of the same kind	fact attached to signature `TRANSITION`	Checked by a “policy” class whenever a transition is made
R16	Goal binding spreading over two working objects	fact `goal_means`	Binding is created by the tool whenever a participation link is created
R17	Means binding for one working object	fact `goal_means`	The localized action to which a means binding can be created by the user must be chosen from a lists of localized actions that relevant to the distributed action from which the means binding will be created
R18	A composition connects two model elements of the same kind	fact attached to signature `COMPOSITION`	Checked by a “policy” class whenever a new model element is created
R19	One composition for a model element and each of its components	fact `cmps_link`	Rendered as line connecting a tree node to each of its child nodes in the tree-view widget

As presented in Chapter 3, the Alloy formalization code has 12 Alloy signatures, 12 Alloy facts (2 of which are unnamed and attached to some Alloy signature), 28 statements, and 130 lines of code. The SeamCAD tool was implemented in standard Java, requiring no special programming libraries other than the standard Java Virtual Machine[11]. The full Java code that implements the SeamCAD computer-aided tool is significantly more complex than the Alloy code that formalizes the SeamCAD modeling language. Figure 30 gives the metrics summary of the latest version of this Java code. It is actually a screenshot of a dialog opened in the SourceMonitor tool[12] - a tool that manages source code written in different programming languages. The total number of lines of Java code is 38002, 37.3% of which are attributed to the comment lines (the

[11] Sun Java Virtual Machine http://java.sun.com/docs/books/jvms/

[12] SourceMonitor http://www.campwoodsw.com/sourcemonitor.html

Java code was documented using JavaDoc[13]). There are a total of 296 Java classes and interfaces. In average, there are 7.2 methods per class, 5.2 statements per method. The total number of Java statements is 16662.



Metrics Summary For Checkpoint 'Joint processes'

Parameter	Value
Project Directory	E:\SeamCAD2Pre\client\src\
Project Name	SeamCAD 2
Checkpoint Name	Joint processes
Created On	28 May 2008, 19:56:33
Files	256
Lines	38'002
Statements	16'662
Percent Branch Statements	14.0
Method Call Statements	9'859
Percent Lines with Comments	37.3
Classes and Interfaces	296
Methods per Class	7.20
Average Statements per Method	5.20
Line Number of Most Complex Method	{undefined}
Name of Most Complex Method	PropertyBoxAsJPanel.setTypeFor()
Maximum Complexity	84
Line Number of Deepest Block	{undefined}
Maximum Block Depth	9+
Average Block Depth	1.81
Average Complexity	2.36

Figure 30. Metrics summary of the Java code that implements the SeamCAD tool

Table 12 compares the metrics of the Alloy code and that of the Java code. It is obvious that the metrics of the Java code is much more enormous than that of the Alloy code due the difference between the two programming paradigms: Alloy is a declarative language and Java is an imperative language. In addition, the Java code implements the user-interface of the SeamCAD tool whereas the Alloy code does not (the execution of the Alloy code is visualized by the Alloy Analyzer tool).

Table 12. Comparison of Alloy code metrics and Java code metrics

Alloy code		Java code	
Number of signatures	12	Number of classes and interfaces	296
fact / signature ratio	1.0	Average number of methods per class	7.2
Number of statements	28	Number of statements	16662
Number of lines of code	130	Number of lines of code	38002
Percentages of comment lines	0%	Percentages of comment lines	37.3%
Number of files	1	Number of files	256

4.5. Data Verification

In this section, we present an efficient way to automatically verify the data manipulated by SeamCAD, for compatibility with the meta-model expressed in Alloy [33]. We check that each instance of the model elements and their relations that is created in the design fulfills the constraints of the meta-model in Alloy. This can be

[13] Sun JavaDoc http://java.sun.com/j2se/javadoc/

efficiently done using CrocoPat[14], a tool for relational programming [34]. First, we export from SeamCAD all model element instances and their relations into a fact base as text file in the relational standard format RSF [35]. Second, we translate the well-formedness rules of the SEAM modeling language that are effectively formalized in Alloy into a relational program in Relational Manipulation Language (RML) - CrocoPat's programming language. The generated RML program contains for each well-formedness rule the corresponding statements to check whether the constraint holds for the whole input fact base (given as RSF file to CrocoPat). If the exported fact base successfully passes the RML program, we are guaranteed that the design as produced by our modeling tool does not break any of the constraints of the meta-model in Alloy.

From Alloy to RML

CrocoPat is an interpreter for imperative, relational programs. By storing the relations internally using a highly tuned representation that is based on binary decision diagrams (BDD), the tool is able to perform complex operations on large relations efficiently. To illustrate the translation from Alloy to CrocoPat, we take a fragment of Alloy code that formalizes the SeamCAD modeling language. A hierarchical element is a generic model element the meta-model. A working object can have a number of child elements and optionally a parent element.

Figure 31 gives the Alloy code fragment. The hierarchical element is declared as a signature and the hierarchy of these elements is constrained by the two Alloy fact `acyclic` (no loop in the hierarchy) and `mutual` (parent and children should be in turn).

```
sig seamHierarchicalElement {
   containment : set seamHierarchicalElement,
   parent : lone seamHierarchicalElement
}

fact acyclic {
   all e: seamHierarchicalElement | e not in (e.^containment + e.^parent)
}

fact mutual {
   all e, epc: seamHierarchicalElement | (epc = e.parent => e in epc.
containment) and (epc in e. containment => epc.parent = e)
}
```

Figure 31. The hierarchical model element is declared in Alloy.

Figure 32 shows the RML (Relational Manipulation Language) code for CrocoPat that is semantically equivalent to the Alloy fact `acyclic` in Figure 31. The Alloy signature `seamHierarchicalElement` is represented by a unary relation (i.e., a set) with the same name; the Alloy fields are represented by binary relations with the same name. The transitive closures are explicitly stored in our example, by the binary relations `TCContainment` and `TCParent`. The fact `acyclic` is represented by an expression whose result is assigned to relation `notAcyclic` ("FA" stands for "for all" in RML). In our relational program, we use the negation of the fact, because we are interested in providing the user with a counterexample if the fact is not valid. Therefore, the RML expression computes the set of objects that *do not* fulfill the constraints of the

[14] CrocoPat: A Tool for Simple and Efficient Relational Programming http://www.cs.sfu.ca/~dbeyer/CrocoPat/

fact `acyclic`. Similarly, fact `mutual` is represented by an expression whose result is assigned to relation `notMutual`. The print statements output the computation results to the standard output. Note that Figure 32 shows only the translation of the two facts. For encoding the cardinality constraint `lone` for the field `parent`, we would have to add the following constraint to the conjunction in Figure 32: `FA(x,y, (parent(e,x) & parent(e,y)) -> x=y)`.

```
TCContainment(x,z) := TC(containment(x,z));
TCParent(x,z)      := TC(parent(x,z));
notAcyclic(e)      :=
  !( seamHierarchicalElement(e) -> ( !TCContainment(e,e) & !TCParent(e,e) ) );
notMutual (e)      :=
 !( seamHierarchicalElement(e) -> FA(c, containment(e,c) -> parent(c,e) &
                                  (parent(e, epc) -> containment(epc, e))
 );

IF( notAcyclic(e) = FALSE(x) ) {
   PRINT "Fact acyclic is valid.", ENDL;
} ELSE {
   PRINT "Fact acyclic is not valid for the following objects:", ENDL;
   PRINT notAcyclic(e);
}
IF( notMutual(e) = FALSE(x) ) {
   PRINT "Fact mutual is valid.", ENDL;
} ELSE {
   PRINT "Fact mutual is not valid for the following objects:", ENDL;
   PRINT notMutual(e);
}
```

Figure 32. An RML program for the Alloy code fragment given in Figure 31.

Figure 33 shows a correct fact base, i.e., relational representation of a design in RSF, for the RML program given in Figure 32. The design consists of four objects: `object0` has no parent but one child, `object1` has `object0` as parent and two children, `object11` and `object12`, which have both `object1` as parent. The unary relation `WorkingObject` contains all objects, while the parent and child relationships are listed by the binary relations `parent` and `containment`, respectively.

CrocoPat is a command-line tool that can be easily integrated into other tools. It gets as input the RML program from Figure 32 and the RSF file from Figure 33, and produces the validity result as ouput. In the negative case it outputs a list of counterexamples.

```
WorkingObject       object0
WorkingObject       object1
WorkingObject       object11
WorkingObject       object12
containment         object0       object1
containment         object1       object11
containment         object1       object12
parent              object1       object0
parent              object11      object1
parent              object12      object1
```

Figure 33. An RSF example describing elements and their relations that is considered as valid design by the RML program

Verification

The Alloy/CrocoPat translator is implemented by integrating the RML code generation into the Alloy parser, using the visitor design pattern. The Java class that translates Alloy to RML is approximately 1400 lines of Java source code. The translation from Alloy to CrocoPat is straightforward for most expressions (both are based on first-order predicate calculus), at the exception of some “syntactic sugar” notations in Alloy. As RML language is based on pure predicate calculus, “abbreviations” in Alloy need to be transformed into compound expressions, i.e., the definition of the notation is inlined. Table 13 summarizes the basic Alloy operators that our prototype implementation can currently translate to RML code.

Table 13. Translation from Alloy to RML
(negation for counterexample extraction included)

<table>
<tr><th>Alloy</th><th>RML</th></tr>
<tr><td colspan="2">Quantified formulas</td></tr>
<tr><td><code><b>all</b> x: sig_A | formula</code></td><td><code>CE(x) := !(sig_A(x)-> <rel expression>)</code></td></tr>
<tr><td><code><b>no</b> x: sig_A | formula</code></td><td><code>CE(x) := sig_A(x) -> <rel expression></code></td></tr>
<tr><td colspan="2">Element formulas</td></tr>
<tr><td><code>y in x.attr</code></td><td><code>attr(x, y)</code></td></tr>
<tr><td><code>x.attr != y.attr</code></td><td><code>!FA(z, attr(x, z) <-> attr(y, z))</code></td></tr>
<tr><td colspan="2">Boolean expressions</td></tr>
<tr><td><code>left_formula && right_formula</code></td><td><code><expression 1> & <expression 2></code></td></tr>
<tr><td><code>left_formula || right_formula</code></td><td><code><expression 1> | <expression 2></code></td></tr>
<tr><td><code>left_formula <=> right_formula</code></td><td><code><expression 1> <-> <expression 2></code></td></tr>
<tr><td colspan="2">Relational expressions</td></tr>
<tr><td><code>left_expr & right_expr</code></td><td><code><rel expression 1> & <rel expression 2></code></td></tr>
<tr><td><code>left_expr | right_expr</code></td><td><code><rel expression 1> | <rel expression 2></code></td></tr>
<tr><td><code>left_expr - right_expr</code></td><td><code><rel expression 1> & !<rel expression 2></code></td></tr>
<tr><td colspan="2">Transitive closure</td></tr>
<tr><td><code>y in x.^attr</code></td><td><code>TC(attr(x, y)</code></td></tr>
</table>

The SeamCAD tool can export the model being edited to a local file in RSF format. As soon as we have RML program that translated from the Alloy code that formalizes the SEAM modeling language, we can run CrocoPat to verify whether the model edited in the SeamCAD tool match the formalization of the SeamCAD modeling language. Figure 34 is the complete RML that was translated from the Alloy code that formalizes the SeamCAD modeling language. Note that the declaration of child and parent elements as well as the related Alloy facts are repeated among the RML code that deal with specific kind of model element.

```
PRINT "========================================================================", ENDL;
PRINT "=====================Verification of SeamCAD model======================", ENDL;
PRINT "========================================================================", ENDL;

PRINT ENDL, "*****Gathering statistics...", ENDL;
PRINT "Number of computational objects: ", #(seamCompuObject(x, y)), ENDL;
PRINT "Number of information objects: ", #(seamInfoObject(x, y)), ENDL;
PRINT "Number of joint actions: ", #(seamJointAction(x, y)), ENDL;
PRINT "Number of localized actions: ", #(seamLocalizedAction(x, y)), ENDL;
PRINT "Number of associations: ", #(ASSOCIATION(x)), ENDL;
PRINT "Number of collaboration links: ", #(COLLABORATION(x)), ENDL;
leafCompuObject(w, n) := seamCompuObject(w, n) & !EX(c, containment(w, c));
PRINT "Number of leaf computational objects: ", #(leafCompuObject(x, y)), ENDL;
mainCompuObject(w, n) := seamCompuObject(w, n) & !EX(p, parent(w, p));
PRINT "Main computational object: ", mainCompuObject(x, y);

//---Signature: seamCompuObject
PRINT ENDL, "*****Checking the cardinalities of 'seamCompuObject'...", ENDL;
```

```
not_lone_main_joint_action(a, n) := seamCompuObject(a, n) & EX(x, y, m, k,
seamJointAction(x, m) & seamJointAction(y, k) & main_joint_action(a, x) &
main_joint_action(a, y) & x != y);
IF (not_lone_main_joint_action(a, n) = FALSE(x)) {
        PRINT "declaration {main_joint_action : lone seamJointAction} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the declaration {main_joint_action : lone
seamJointAction}", ENDL;
        PRINT not_lone_main_joint_action(a, n);
}

not_lone_main_property(a, n) := seamCompuObject(a, n) & EX(x, y, m, k, seamInfoObject(x,
m) & seamInfoObject(y, k) & main_property(a, x) & main_property(a, y) & x != y);
IF (not_lone_main_property(a, n) = FALSE(x)) {
        PRINT "declaration {main_property : lone seamInfoObject} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the declaration {main_property : lone
seamInfoObject}", ENDL;
        PRINT not_lone_main_property(a, n);
}

not_lone_main_localized_action(a, n) := seamCompuObject(a, n) & EX(x, y, m, k,
seamLocalizedAction(x, m) & seamLocalizedAction(y, k) & main_localized_action(a, x) &
main_localized_action(a, y) & x != y);
IF (not_lone_main_localized_action(a, n) = FALSE(x)) {
        PRINT "declaration {main_localized_action : lone seamLocalizedAction} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the declaration {main_localized_action : lone
seamLocalizedAction}", ENDL;
        PRINT not_lone_main_localized_action(a, n);
}

compu_not_lone_parent(a, n) := seamCompuObject(a, n) & EX(x, y, m, k, seamCompuObject(x,
m) & seamCompuObject(y, k) & parent(a, x) & parent(a, y) & x != y);
IF (compu_not_lone_parent(a, n) = FALSE(x)) {
        PRINT "declaration {parent: lone seamCompuObject} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the declaration {parent: lone
seamCompuObject}", ENDL;
        PRINT compu_not_lone_parent(a, n);
}

//---Signature: seamInfoObject
PRINT ENDL, "*****Checking the cardinalities of 'seamInfoObject'...", ENDL;

info_not_one_compu_host(a, n) := seamInfoObject(a, n) & !(EX(x, m, seamCompuObject(x, m)
& compu_host(a, x)) & !EX(x, y, m, k, seamCompuObject(x, m) & seamCompuObject(y, k) &
compu_host(a, x) & compu_host(a, y) & x != y));
IF (info_not_one_compu_host(a, n) = FALSE(x)) {
        PRINT "declaration {compu_host : one seamCompuObject} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the declaration {one compu_host}", ENDL;
        PRINT info_not_one_compu_host(a, n);
}

info_not_lone_parent(a, n) := seamInfoObject(a, n) & EX(x, y, m, k, seamInfoObject(x, m)
& seamInfoObject(y, k) & parent(a, x) & parent(a, y) & x != y);
IF (info_not_lone_parent(a, n) = FALSE(x)) {
        PRINT "declaration {parent: lone seamInfoObject} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the declaration {parent: lone seamInfoObject}",
ENDL;
        PRINT info_not_lone_parent(a, n);
}

//---Signature: seamJointAction
PRINT ENDL, "*****Checking the cardinalities of 'seamJointAction'...", ENDL;

joint_not_one_compu_host(a, n) := seamJointAction(a, n) & !(EX(x, m, seamCompuObject(x,
m) & compu_host(a, x)) & !EX(x, y, m, k, seamCompuObject(x, m) & seamCompuObject(y, k) &
compu_host(a, x) & compu_host(a, y) & x != y));
```

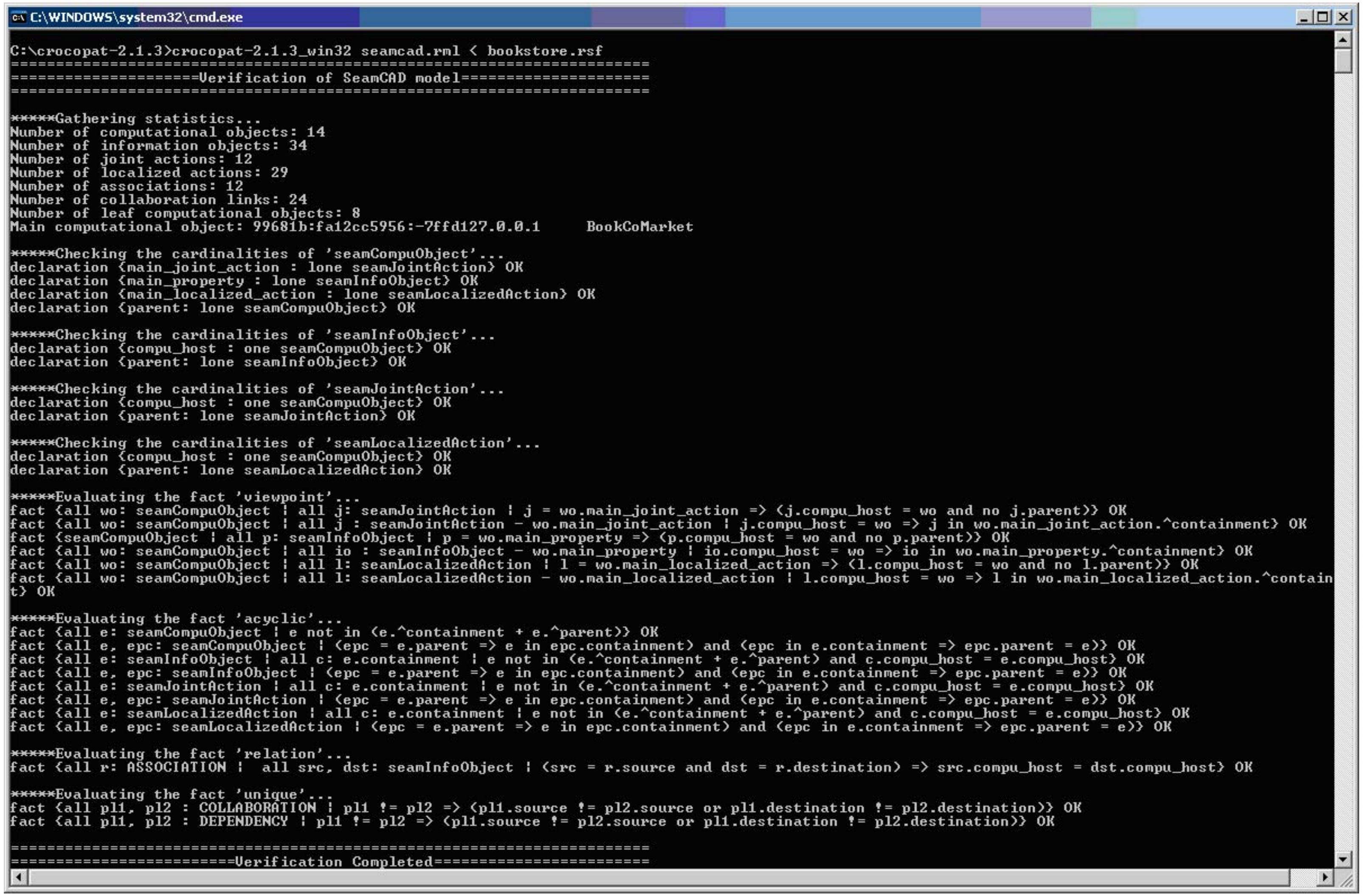
```
IF (joint_not_one_compu_host(a, n) = FALSE(x)) {
	PRINT "declaration {compu_host : one seamCompuObject} OK", ENDL;
}
ELSE {
	PRINT "Element(s) that violate(s) the declaration {compu_host : one
seamCompuObject}", ENDL;
	PRINT joint_not_one_compu_host(a, n);
}

joint_not_lone_parent(a, n) := seamJointAction(a, n) & EX(x, y, m, k, seamJointAction(x,
m) & seamJointAction(y, k) & parent(a, x) & parent(a, y) & x != y);
IF (joint_not_lone_parent(a, n) = FALSE(x)) {
	PRINT "declaration {parent: lone seamJointAction} OK", ENDL;
}
ELSE {
	PRINT "Element(s) that violate(s) the declaration {parent: lone
seamJointAction}", ENDL;
	PRINT joint_not_lone_parent(a, n);
}

//---Signature: seamLocalizedAction
PRINT ENDL, "*****Checking the cardinalities of 'seamLocalizedAction'...", ENDL;

localized_not_one_compu_host(a, n) := seamLocalizedAction(a, n) & !(EX(x, m,
seamCompuObject(x, m) & compu_host(a, x)) & !EX(x, y, m, k, seamCompuObject(x, m) &
seamCompuObject(y, k) & compu_host(a, x) & compu_host(a, y) & x != y));
IF (localized_not_one_compu_host(a, n) = FALSE(x)) {
	PRINT "declaration {compu_host : one seamCompuObject} OK", ENDL;
}
ELSE {
	PRINT "Element(s) that violate(s) the declaration {compu_host : one
seamCompuObject}", ENDL;
	PRINT localized_not_one_compu_host(a, n);
}

localized_not_lone_parent(a, n) := seamLocalizedAction(a, n) & EX(x, y, m, k,
seamLocalizedAction(x, m) & seamLocalizedAction(y, k) & parent(a, x) & parent(a, y) & x
!= y);
IF (localized_not_lone_parent(a, n) = FALSE(x)) {
	PRINT "declaration {parent: lone seamLocalizedAction} OK", ENDL;
}
ELSE {
	PRINT "Element(s) that violate(s) the declaration {lone parent}", ENDL;
	PRINT localized_not_lone_parent(a, n);
}

//---Facts:
//Fact: fact viewpoint
PRINT ENDL, "*****Evaluating the fact 'viewpoint'...", ENDL;

DescendantOf(x, z) := TC(containment(x, z));
AncestorOf(x, z) := TC(parent(x, z));

not_viewpoint1(wo, n) := seamCompuObject(wo, n) & !FA(j, main_joint_action(wo, j) ->
(compu_host(j, wo) & !EX(y, parent(j, y) ) ) );
IF (not_viewpoint1(wo, n) = FALSE(x)) {
	PRINT "fact {all wo: seamCompuObject | all j: seamJointAction | j =
wo.main_joint_action => (j.compu_host = wo and no j.parent)} OK", ENDL;
}
ELSE {
	PRINT "Element(s) that violate(s) the fact {all wo :  seamCompuObject | all j :
seamJointAction | some wo . main_joint_action && j = wo . main_joint_action => j .
compu_host = wo && no j . parent}", ENDL;
	PRINT not_viewpoint1(wo, n);
}

not_viewpoint2(wo, n) := seamCompuObject(wo, n) & EX(k, main_joint_action(wo, k)) &
!EX(k, main_joint_action(wo, k) & FA(j, m, seamJointAction(j, m) &
!main_joint_action(wo, j) & compu_host(j, wo)-> DescendantOf(k, j)));
IF ( not_viewpoint2(wo, n) = FALSE(x) ) {
	PRINT "fact {all wo: seamCompuObject | all j : seamJointAction -
wo.main_joint_action | j.compu_host = wo => j in wo.main_joint_action.^containment} OK",
ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all wo: seamCompuObject | all j :
seamJointAction - wo.main_joint_action | j.compu_host = wo => j in
wo.main_joint_action.^containment}", ENDL;
        PRINT not_viewpoint2(wo, n);
}

not_viewpoint3(wo, n) := seamCompuObject(wo, n) & !FA(p, m, seamInfoObject(p, m) &
main_property(wo, p) -> (compu_host(p, wo) & !EX(y, parent(p, y) ) ) );
IF ( not_viewpoint3(wo, n) = FALSE(x) ) {
        PRINT "fact {seamCompuObject | all p: seamInfoObject | p = wo.main_property =>
(p.compu_host = wo and no p.parent)} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all wo: seamCompuObject | all p:
seamInfoObject | p = wo.main_property => (p.compu_host = wo and no p.parent)}", ENDL;
        PRINT not_viewpoint3(wo, n);
}

not_viewpoint4(wo, n) := seamCompuObject(wo, n) & EX(k, main_property(wo, k)) & !EX(k,
main_property(wo, k) & FA(p, m, seamInfoObject(p, m) & !main_property(wo, p) &
compu_host(p, wo)-> DescendantOf(k, p)));
IF ( not_viewpoint4(wo, n) = FALSE(x) ) {
        PRINT "fact {all wo: seamCompuObject | all io : seamInfoObject - wo.main_property
| io.compu_host = wo => io in wo.main_property.^containment} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all wo: seamCompuObject | all io :
seamInfoObject - wo.main_property | io.compu_host = wo => io in
wo.main_property.^containment}", ENDL;
        PRINT not_viewpoint4(wo, n);
}

not_viewpoint5(wo, n) := seamCompuObject(wo, n) & !FA(l, main_localized_action(wo, l) ->
(compu_host(l, wo) & !EX(y, parent(l, y) ) ) );
IF ( not_viewpoint5(wo, n) = FALSE(x) ) {
        PRINT "fact {all wo: seamCompuObject | all l: seamLocalizedAction | l =
wo.main_localized_action => (l.compu_host = wo and no l.parent)} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact all wo: seamCompuObject | all l:
seamLocalizedAction | l = wo.main_localized_action => (l.compu_host = wo and no
l.parent)}", ENDL;
        PRINT not_viewpoint5(wo, n);
}

not_viewpoint6(wo, n) := seamCompuObject(wo, n) & EX(k, main_localized_action(wo, k)) &
!EX(k, main_localized_action(wo, k) & FA(l, m, seamLocalizedAction(l, m) &
!main_localized_action(wo,l) & compu_host(l, wo)-> DescendantOf(k, l)));
IF ( not_viewpoint6(wo, n) = FALSE(x) ) {
        PRINT "fact {all wo: seamCompuObject | all l: seamLocalizedAction -
wo.main_localized_action | l.compu_host = wo => l in
wo.main_localized_action.^containment} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all wo: seamCompuObject | all l:
seamLocalizedAction - wo.main_localized_action | l.compu_host = wo => l in
wo.main_localized_action.^containment}", ENDL;
        PRINT not_viewpoint6(wo, n);
}

//Fact: fact acyclic
PRINT ENDL, "*****Evaluating the fact 'acyclic'...", ENDL;

compu_not_acyclic(e, n) := !(seamCompuObject(e, n) -> !DescendantOf(e, e) &
!AncestorOf(e, e));
IF ( compu_not_acyclic(e, n) = FALSE(x) ) {
        PRINT "fact {all e: seamCompuObject | e not in (e.^containment + e.^parent)} OK",
ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all e: seamCompuObject | e not in
(e.^containment + e.^parent)}", ENDL;
        PRINT compu_not_acyclic(e, n);
}

compu_not_containment(e, n) := !(seamCompuObject(e, n) -> FA(epc, (containment(e, epc) -
> parent(epc, e)) & (parent(e, epc) -> containment(epc, e))));
```

```
IF ( compu_not_containment(e, n) = FALSE(x) ) {
       PRINT "fact {all e, epc: seamCompuObject | (epc = e.parent => e in
epc.containment) and (epc in e.containment => epc.parent = e)} OK", ENDL;
}
ELSE {
       PRINT "Element(s) that violate(s) the fact {all e, epc: seamCompuObject | (epc =
e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)}", ENDL;
       PRINT compu_not_containment(e, n);
}

info_not_acyclic(e, n) := !(seamInfoObject(e, n) -> !DescendantOf(e, e) & !AncestorOf(e,
e) & FA(c, containment(e, c) -> EX(host, compu_host(c, host) & compu_host(e, host))));
IF( info_not_acyclic(e, n) = FALSE(x) ) {
       PRINT "fact {all e: seamInfoObject | all c: e.containment | e not in
(e.^containment + e.^parent) and c.compu_host = e.compu_host} OK", ENDL;
}
ELSE {
       PRINT "Element(s) that violate(s) the fact {all e: seamInfoObject | all c:
e.containment | e not in (e.^containment + e.^parent) and c.compu_host = e.compu_host}",
ENDL;
       PRINT info_not_acyclic(e, n);
}

info_not_containment(e, n) := !(seamInfoObject(e, n) -> FA(epc, (containment(e, epc) ->
parent(epc, e)) & (parent(e, epc) -> containment(epc, e))));
IF ( info_not_containment(e, n) = FALSE(x) ) {
       PRINT "fact {all e, epc: seamInfoObject | (epc = e.parent => e in
epc.containment) and (epc in e.containment => epc.parent = e)} OK", ENDL;
}
ELSE {
       PRINT "Element(s) that violate(s) the fact {all e, epc: seamInfoObject | (epc =
e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)}", ENDL;
       PRINT info_not_containment(e, n);
}

joint_not_acyclic(e, n) := !(seamJointAction(e, n) -> !DescendantOf(e, e) &
!AncestorOf(e, e) & FA(c, containment(e, c) -> EX(host, compu_host(c, host) &
compu_host(e, host))));
IF( joint_not_acyclic(e, n) = FALSE(x) ) {
       PRINT "fact {all e: seamJointAction | all c: e.containment | e not in
(e.^containment + e.^parent) and c.compu_host = e.compu_host} OK", ENDL;
}
ELSE {
       PRINT "Element(s) that violate(s) the fact {all e: seamJointAction | all c:
e.containment | e not in (e.^containment + e.^parent) and c.compu_host = e.compu_host}",
ENDL;
       PRINT joint_not_acyclic(e, n);
}

joint_not_containment(e, n) := !(seamJointAction(e, n) -> FA(epc, (containment(e, epc) -
> parent(epc, e)) & (parent(e, epc) -> containment(epc, e))));
IF ( joint_not_containment(e, n) = FALSE(x) ){
       PRINT "fact {all e, epc: seamJointAction | (epc = e.parent => e in
epc.containment) and (epc in e.containment => epc.parent = e)} OK", ENDL;
}
ELSE {
       PRINT "Element(s) that violate(s) the fact {all e, epc: seamJointAction | (epc =
e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)}", ENDL;
       PRINT joint_not_containment(e, n);
}

localized_not_acyclic(e, n) := !(seamLocalizedAction(e, n) -> !DescendantOf(e, e) &
!AncestorOf(e, e) & FA(c, containment(e, c) -> EX(host, compu_host(c, host) &
compu_host(e, host))));
IF( localized_not_acyclic(e, n) = FALSE(x) ){
       PRINT "fact {all e: seamLocalizedAction | all c: e.containment | e not in
(e.^containment + e.^parent) and c.compu_host = e.compu_host} OK", ENDL;
}
ELSE {
       PRINT "Element(s) that violate(s) the fact {all e: seamLocalizedAction | all c:
e.containment | e not in (e.^containment + e.^parent) and c.compu_host = e.compu_host}",
ENDL;
       PRINT localized_not_acyclic(e, n);
}

localized_not_containment(e, n) := !(seamLocalizedAction(e, n) -> FA(epc,
(containment(e, epc) -> parent(epc, e)) & (parent(e, epc) -> containment(epc, e))));
```

```
IF ( localized_not_containment(e, n) = FALSE(x) ){
        PRINT "fact {all e, epc: seamLocalizedAction | (epc = e.parent => e in
epc.containment) and (epc in e.containment => epc.parent = e)} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all e, epc: seamLocalizedAction |
(epc = e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)}",
ENDL;
        PRINT localized_not_containment(e, n);
}

//Fact: fact relation
PRINT ENDL, "*****Evaluating the fact 'relation'...", ENDL;
crossing_ass(r) := ASSOCIATION(r) & !FA(src, dst, n, m, seamInfoObject(src, n) &
seamInfoObject(dst, m) & source(r, src) & destination(r, dst) -> EX(host,
compu_host(src, host) & compu_host(dst, host)));
IF( crossing_ass(e) != FALSE(x) ) {
  PRINT "fact {all r: ASSOCIATION |  all src, dst: seamInfoObject | (src = r.source and
dst = r.destination) => src.compu_host = dst.compu_host} is not satisfied", ENDL, "all
these objects violate the fact:", ENDL;
  PRINT crossing_ass(e);
}
ELSE { PRINT "fact {all r: ASSOCIATION |  all src, dst: seamInfoObject | (src = r.source
and dst = r.destination) => src.compu_host = dst.compu_host} OK", ENDL; }

//Fact: fact unique
PRINT ENDL, "*****Evaluating the fact 'unique'...", ENDL;

col_not_unique(r) := COLLABORATION(r) & !FA(rx, COLLABORATION(rx) & r != rx -> EX(src1,
src2, source(r, src1) & source(rx, src2) & src1 != src2) | EX(dst1, dst2, destination(r,
dst1) & destination(rx, dst2) & dst1 != dst2));
IF (col_not_unique(e) = FALSE(x)) {
        PRINT "fact {all pl1, pl2 : COLLABORATION | pl1 != pl2 => (pl1.source !=
pl2.source or pl1.destination != pl2.destination)} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all pl1, pl2 : COLLABORATION | pl1 !=
pl2 => (pl1.source != pl2.source or pl1.destination != pl2.destination)}", ENDL;
        PRINT col_not_unique(e);
}

dep_not_unique(r) := DEPENDENCY(r) & !FA(rx, DEPENDENCY(rx) & r != rx -> EX(src1, src2,
source(r, src1) & source(rx, src2) & src1 != src2) | EX(dst1, dst2, destination(r, dst1)
& destination(rx, dst2) & dst1 != dst2));
IF (dep_not_unique(e) = FALSE(x)) {
        PRINT "fact {all pl1, pl2 : DEPENDENCY | pl1 != pl2 => (pl1.source != pl2.source
or pl1.destination != pl2.destination)} OK", ENDL;
}
ELSE {
        PRINT "Element(s) that violate(s) the fact {all pl1, pl2 : DEPENDENCY | pl1 !=
pl2 => (pl1.source != pl2.source or pl1.destination != pl2.destination)} is not
satisfied", ENDL;
        PRINT dep_not_unique(e);
}

PRINT ENDL, "=======================================================================",
ENDL;
PRINT "=========================Verification Completed========================", ENDL;
PRINT "=======================================================================", ENDL;
```

Figure 34. Complete RML code that verifies the instances of enterprise models edited in the SeamCAD tool

Figure 35 is the screenshot of running CrocoPat having the RSF data of the bookstore model and the translated RML code as inputs. We can see that all the RML statements that were translated from the formalization code of the SeamCAD modeling language are respected.



```
C:\crocopat-2.1.3>crocopat-2.1.3_win32 seamcad.rml < bookstore.rsf
======================================================================
===================Verification of SeamCAD model=====================
======================================================================

*****Gathering statistics...
Number of computational objects: 14
Number of information objects: 34
Number of joint actions: 12
Number of localized actions: 29
Number of associations: 12
Number of collaboration links: 24
Number of leaf computational objects: 8
Main computational object: 99681b:fa12cc5956:-7ffd127.0.0.1      BookCoMarket

*****Checking the cardinalities of 'seamCompuObject'...
declaration {main_joint_action : lone seamJointAction} OK
declaration {main_property : lone seamInfoObject} OK
declaration {main_localized_action : lone seamLocalizedAction} OK
declaration {parent: lone seamCompuObject} OK

*****Checking the cardinalities of 'seamInfoObject'...
declaration {compu_host : one seamCompuObject} OK
declaration {parent: lone seamInfoObject} OK

*****Checking the cardinalities of 'seamJointAction'...
declaration {compu_host : one seamCompuObject} OK
declaration {parent: lone seamJointAction} OK

*****Checking the cardinalities of 'seamLocalizedAction'...
declaration {compu_host : one seamCompuObject} OK
declaration {parent: lone seamLocalizedAction} OK

*****Evaluating the fact 'viewpoint'...
fact {all wo: seamCompuObject | all j: seamJointAction | j = wo.main_joint_action => (j.compu_host = wo and no j.parent)} OK
fact {all wo: seamCompuObject | all j : seamJointAction - wo.main_joint_action | j.compu_host = wo => j in wo.main_joint_action.^containment} OK
fact {seamCompuObject | all p: seamInfoObject | p = wo.main_property => (p.compu_host = wo and no p.parent)} OK
fact {all wo: seamCompuObject | all io : seamInfoObject - wo.main_property | io.compu_host = wo => io in wo.main_property.^containment} OK
fact {all wo: seamCompuObject | all l: seamLocalizedAction | l = wo.main_localized_action => (l.compu_host = wo and no l.parent)} OK
fact {all wo: seamCompuObject | all l: seamLocalizedAction - wo.main_localized_action | l.compu_host = wo => l in wo.main_localized_action.^contain
t} OK

*****Evaluating the fact 'acyclic'...
fact {all e: seamCompuObject | e not in (e.^containment + e.^parent)} OK
fact {all e, epc: seamCompuObject | (epc = e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)} OK
fact {all e: seamInfoObject | all c: e.containment | e not in (e.^containment + e.^parent) and c.compu_host = e.compu_host} OK
fact {all e, epc: seamInfoObject | (epc = e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)} OK
fact {all e: seamJointAction | all c: e.containment | e not in (e.^containment + e.^parent) and c.compu_host = e.compu_host} OK
fact {all e, epc: seamJointAction | (epc = e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)} OK
fact {all e: seamLocalizedAction | all c: e.containment | e not in (e.^containment + e.^parent) and c.compu_host = e.compu_host} OK
fact {all e, epc: seamLocalizedAction | (epc = e.parent => e in epc.containment) and (epc in e.containment => epc.parent = e)} OK

*****Evaluating the fact 'relation'...
fact {all r: ASSOCIATION |  all src, dst: seamInfoObject | (src = r.source and dst = r.destination) => src.compu_host = dst.compu_host} OK

*****Evaluating the fact 'unique'...
fact {all pl1, pl2 : COLLABORATION | pl1 != pl2 => (pl1.source != pl2.source or pl1.destination != pl2.destination)} OK
fact {all pl1, pl2 : DEPENDENCY | pl1 != pl2 => (pl1.source != pl2.source or pl1.destination != pl2.destination)} OK

======================================================================
========================Verification Completed========================
```

Figure 35. Verification result yielded by Crocopat for
the model of the online bookstore edited in SeamCAD tool

Chapter 5: Applications and Feedback

***<u>Overview</u>*:** *This chapter presents applications and feedback on SeamCAD. We applied SeamCAD in several projects, some of which were in conjunction with industry. An enterprise model was built using SeamCAD for the case-study of a master's course given by our group. This case-study is about a company that manufactures, sells and maintains lightweight aircraft engines. An application was made in a company that wanted to build an enterprise model to manage its sale processes and its customer relations in the market of watch-parts manufacturing. Another project used SeamCAD in making an enterprise model for a new department building on our university campus. This model was useful for specifying how the building should be equipped and what IT system should be installed in a new building of the school. An additional project was set up to investigate the possibility to further the model created in SeamCAD to be able to simulate System Dynamics. The feedback on SeamCAD from 20 people, including practitioners and researchers in the fields related to Enterprise Architecture and our master's students are also presented in this chapter.*

5.1. Applications

In the research group where this Ph.D. work was carried out, SeamCAD was applied in several projects some of which were in conjunction with industry.

5.1.1. A case-study enterprise model in a master's course on EA and SOA

In our group, a course is given to master's students to teach them how to build up a company that manufactures and sells through a game case-study [36]. In this course, students are asked to make an enterprise model for their imaginary company. They are divided into groups of four to six. Each group represents a company called, for example BE (Best Engines SA), which manufactures and sells diesel-powered engines for lightweight aircrafts. The company can buy parts and design from suppliers and manages its own inventory. The companies represented by students groups should compete with one another to sell engines they manufacture to a company called, for instance NewPlane SA, which itself sells aircrafts to Dawa (DAnce With the Angels) - an air club who makes business on lightweight aircrafts (rental, training pilot…). In addition, the companies run by the students also have common competitors: companies that manufactures and sells gas-powered engines for lightweight aircrafts. These companies in fact have more market share because traditionally lightweight aircrafts are powered by gas. However, as diesel-powered engines are developed using modern-day technologies, they are more economically operating and at the same time more difficult to maintain.

In this problem-based course, the students are asked to improve the way their companies maintain the engines they sold to the air club (via NewPlane SA). When an air club member brings an aircraft that no longer functions correctly to the reception of the air club, the local garage of the air club can make an initial diagnosis on the aircraft engine. If the engine is broken, the BE company is contacted to solve the problem. It needs a replacing part a trained technician (i.e. one who is certified for repairing diesel-powered engines) to fix the problematic engine. Due to the inefficiency of the current telephone- and paper-based communication between the local garage of Dawa, the special garage that manages certified technicians and the BE company, the whole reparation process for broken aircrafts is often unnecessarily delayed. The students need to develop an IT system for their companies that can handle this communication in a much more efficient way.

The enterprise model of the company BE can be built in a hierarchical way. In this section, the enterprise model built using SeamCAD is presented. All model elements are expressed with the building blocks of the SeamCAD modeling language and the entire enterprise model of the BE company can efficiently be browsed and easily understood thanks to the SeamCAD computer-aided tool. This enterprise model can serve as a sample model for students. They can refer to it after they have created an enterprise model of their own BE company.

Figure 36 shows a diagram that represents the first organizational level. A market of lightweight aircraft has two segments: diesel-powered engine for light aircrafts and gas-powered engine for light aircrafts. Both of them have the same interest which is to grow their own market share.

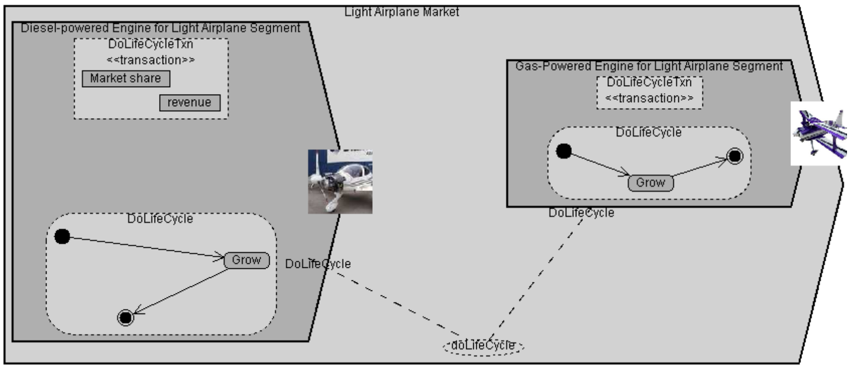


Figure 36. The 1st organizational level showing
a market of airplane engines that has two segments

Figure 37 shows another diagram that represents the second organizational level. In the segment of diesel-powered engine for light aircrafts, there are two value networks: value network of the BE company (`BE Value Network`) and value network of the air club (`Dawa Value Network`). Note that this diagram shows the second functional level, not the first one. The two value networks conduct sale of engines and recycle them at the end of their lifecycle (distributed action `sale` and `recycle`). The `BE value network` also takes part in managing engines that are in operation at `Dawa Value Network` (distributed action `manageAtDawa`). The overall collaboration between the two value networks deal with the entire lifecycle of diesel-powered engines and is expressed a distributed action called `engineLifecycle`. A typical order of business processes at `BE Value Network` is: to manufacture engines, to get involved in managing engines that are in operation at `Dawa Value Network` and to recycle dead engines. A typical order of business processes at `Dawa Value network` is: to install engines, to manage engines that are in operation and to recycle dead engines.

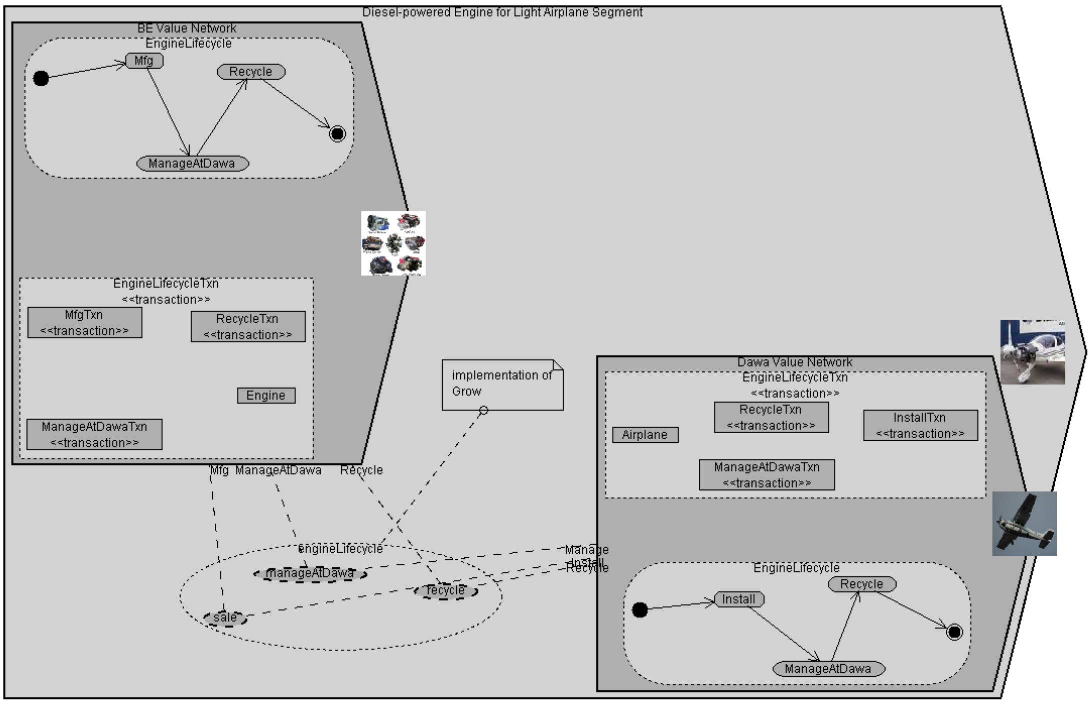


Figure 37. The 2nd organizational level and the 2nd functional level
showing the two value networks of the `BE` and `Dawa`.

The diagram shown in Figure 38 also represents the same organizational level of the BE enterprise model as Figure 37 but a more detailed functional level. In fact, the localized actions `ManageAtDawa` of the two value networks are detailed in this diagram. Both the `BE value network` and `Dawa Value network` takes part in the `repair` distributed action. `Dawa Value network` operate, maintain engines in their aircrafts but `BE value network` does not. A set of all possible transitions between the three localized actions `Operate`, `Maintain` and `Repair` implies that `Dawa Value network` can them in any order.

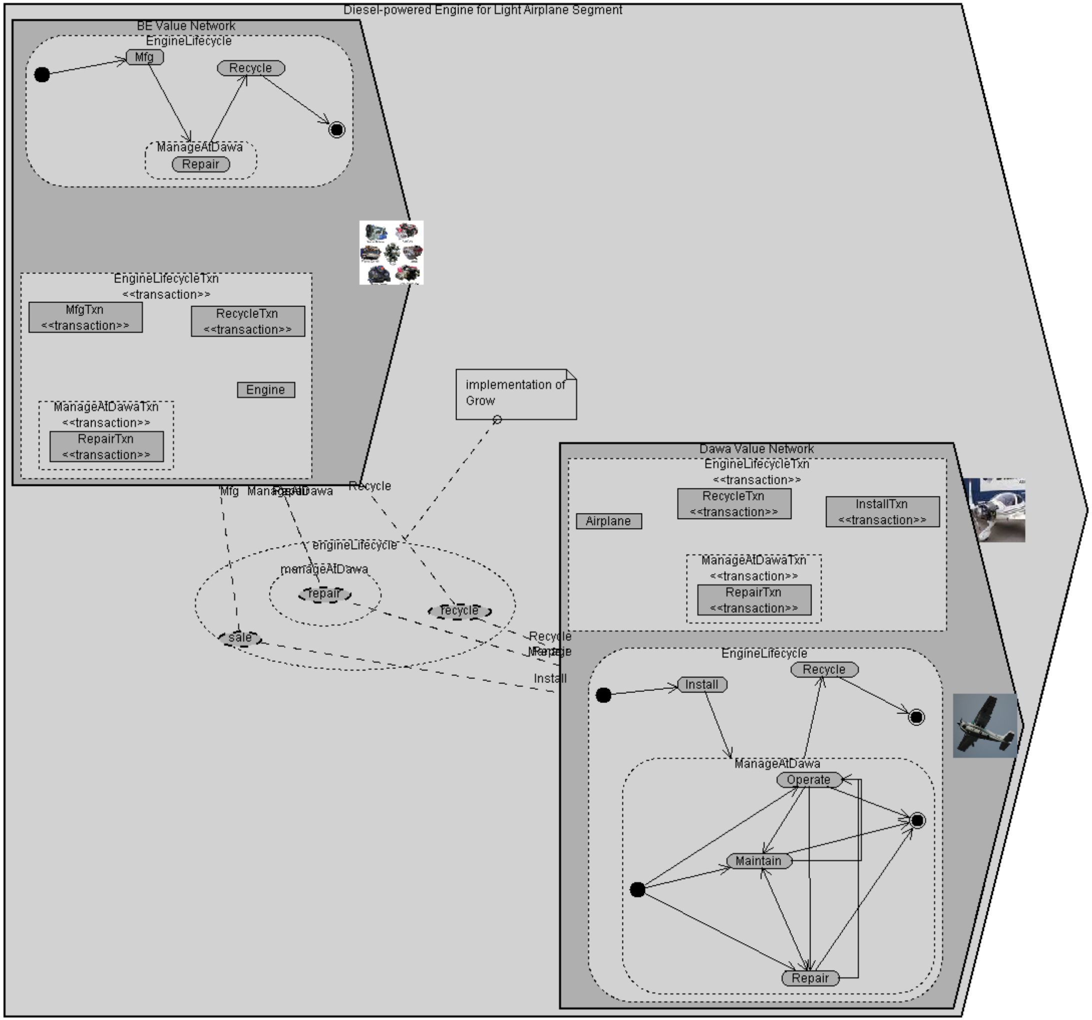


Figure 38. The 2nd organizational level and the 3rd functional level showing the two value networks of the `BE` and `Dawa` doing `repair`.

In Figure 39, the distributed action `repair` is detailed. Accordingly, the two localized actions `Repair` and the properties related to these localized actions are also detailed. The distributed action `repair` is broken down into 5 component actions that specifically do: receiving a problematic aircraft from a member of the air club, examining the potentially-broken engine of the received aircraft, ordering parts that necessary to repair the broken engine, calling for a certified technician who is able to replace broken parts in the engine, mounting the repaired engine to the aircraft and returning it to the air club member.

For `BE Value Network`, the component localized actions of the localized action `Repair` are: begin the reparation, get diagnosis result (from Dawa), deliver parts needed, send a certified technician (to Dawa) and end the reparation. These localized actions are listed in the order they happen.

For `Dawa Value Network`, the component localized actions of the localized action `Repair` are: reception of an aircraft that needs to be repaired from member, diagnose the engine of the aircraft received, analyze the diagnosis results, order the replacing parts that are necessary to fix the broken engine, call a certified technician (from BE value network) who can do part replacement, parts are replaced and the aircraft is returned to the member. These localized actions are listed in the order they happen.

As we can see in Figure 39, information is exchanged between the two value networks during the reparation process. In the stateless property `RepairTxn` that represents the occurrence of the localized action `Repair` of both the `BE Value Network` and the `Dawa Value Network`, there are stateful properties that represent information related to the reparation process. Most of them have stereotype `<<in>>` or `<<out>>`, which indicate that they are either input or output to the `RepairTxn` where they are defined. First, in the stateless property `RepairTxn` of `Dawa Value Network`, `Airplane KO` indicates that the engine of the aircraft received from the member is problematic. On the side of `BE Value Network`, the stateful property `reparation request` is an input property that is traceable from `Airplane KO`. On the side of `Dawa Value Network`, `diagnosis result` is an output property. The corresponding input property on the side of `BE Value Network` is `diagnosis report`. There are also two output properties, `technician` and `spare part`, that represent the technician and the replacing parts sent to Dawa. On the side of the `Dawa Value Network`, two corresponding input properties are `expert` and `replacing part`. Finally, an output property `Airplane OK` is placed on the side of `Dawa Value Network` and its corresponding input property on the side of `BE Value Network` is `approval`.

Some transitions between component actions of the localized action `Repair` of the two value networks are associated with conditions that imply when input or output properties are ready so to fire the transitions. If an input property is specified for a transition condition, the transition is fired upon the reception of the property. If an output property is specified for a transition condition, the transition is fired when the property is sent. For instance, once the property `approval` is received in the `RepairTxn` of `BE Value Network`, the transition from the localized action `Send technician` to the localized action `End repairing` is fired. When the property `diagnosis result` is sent from the `RepairTxn` of `Dawa Value Network`, the transition from the localized action `Diagnose` to the localized action `Analyze` is fired.

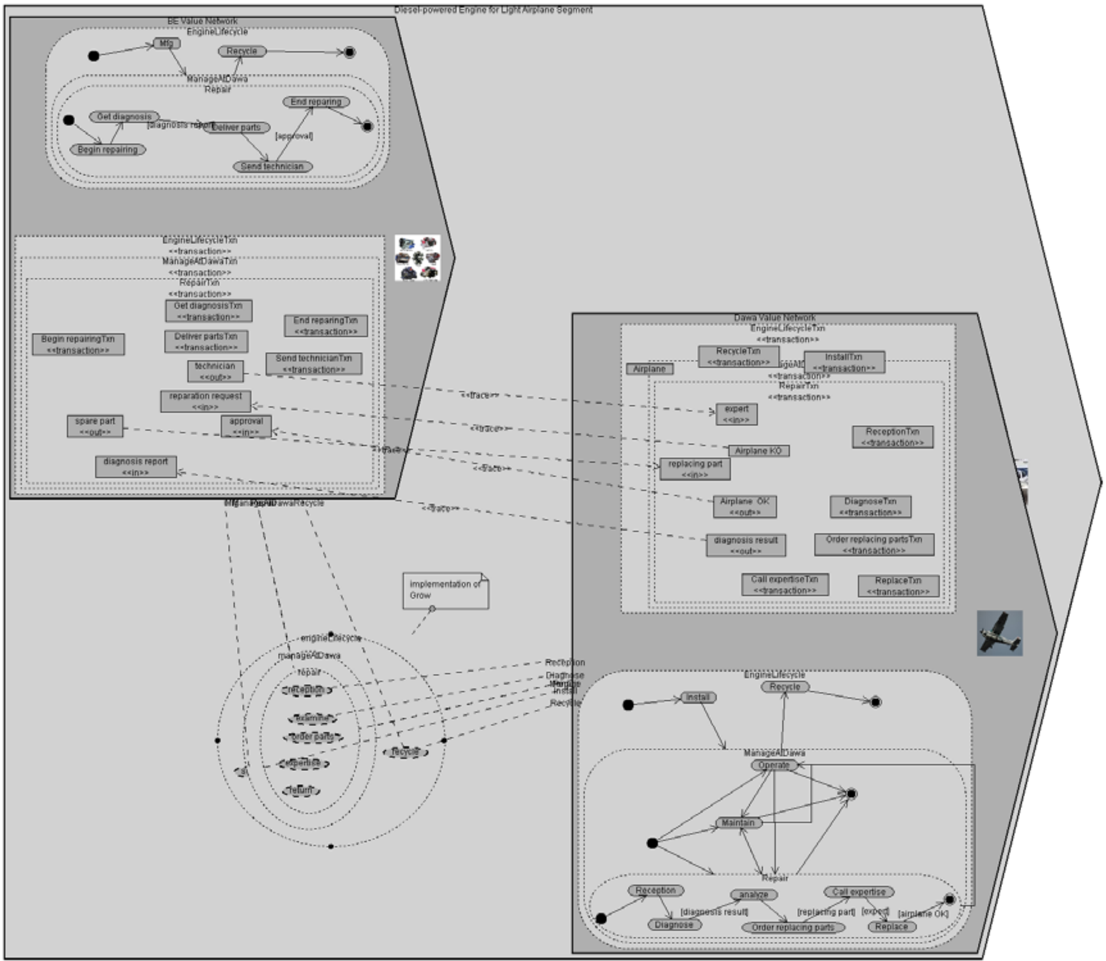


Figure 39. The 2nd organizational level and the 4rh functional level showing the two value networks of the `BE` and `Dawa` doing `repair`.

In Figure 40, the `BE Value Network` is seen as composite. This value network has an IT system that manages the reparation data (e.g. delivery time of the replacing parts, schedule for a certified technician to do reparation at Dawa), a delivery company called `DeliveryCo` that is responsible for delivering the replacing parts, a garage called `All Can Do` that manages certified technicians and an agent `OFAC` who can certify technicians. They are all working objects that collaborate to implement the services exposed by the `BE Value Network` as localized actions that are visible in Figure 39. First of all, the IT system gets necessary information about the reparation (e.g. name of the member whose aircraft needs repairing). Then it proposes the delivery time of replacing parts based on their availability. `DeliveryCo` delivers the parts within the proposed delivery time. `All Can Do` uses this IT system to appoint a certified technician. This garage then sends the technician. The IT system finally gets some notification informing that the reparation process has been done. It validates it and store the information about the entire reparation process in its database.

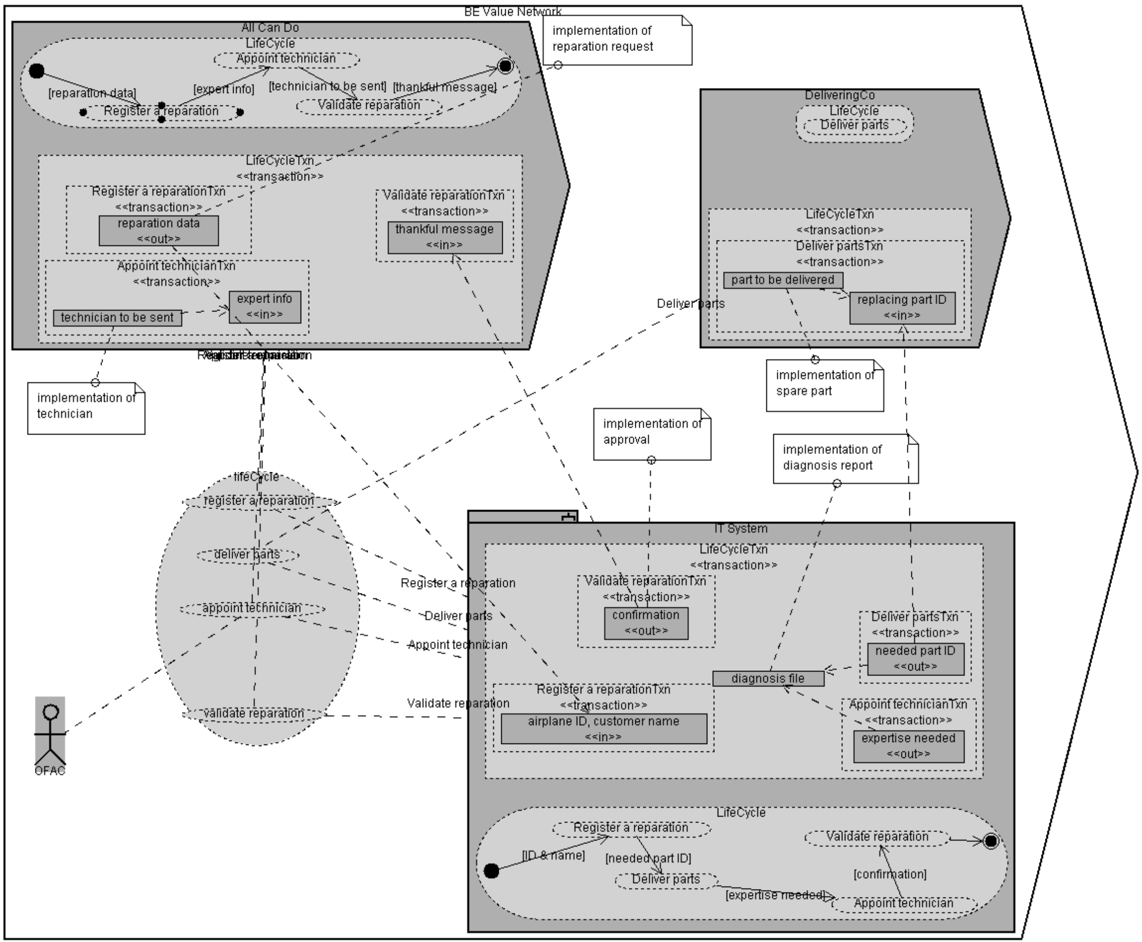


Figure 40. The 3rd organizational level and 2nd functional level showing how the IT system, the garage and the delivering company collaborate to implement the reparation.

The enterprise model created in this case-study has a total of 3 organizational levels. In this enterprise model, the deepest functional level can be observed in the representation of the segment of diesel-powered aircrafts (see Figure 39). After having created the enterprise model in the SeamCAD tool, it is possible to browse it and open all diagrams illustrated in this subsection. Using the tool, the student would understand the notion of hierarchy and the building blocks of the SeanCAD modeling language quickly. As the SeamCAD modeling language was based on the SEAM method [3] [4] [5], the students would also learnt the modeling terms that are introduced in the later stage of the course more efficiently.

It is possible to make BPMN diagrams out of this enterprise model although the SeamCAD tool does not support this feature. For example, the diagram shown in Figure 39 can straightforwardly be mapped to a BMMN diagram by doing the following steps

1. `BE Value Network` and `Dawa Value Network` are mapped to pools
2. All the localized actions of `BE Value Network` become processes in the pool that represents this value network
3. All the localized actions of `Dawa Value Network` become processes in the pool that represents this value network
4. Each transition is mapped to a sequence flow
5. Each pair of input / output properties may be mapped to a data object.
6. Each line denoting a trace relationship is mapped to a message flow

5.1.2. Enterprise model of an ERP-seeking company in the market of watch parts manufacturing

A company that is active in the development of ERP (Enterprise Resource Planning) solutions and the research of a method for representing the customers' needs in the market of watch manufacturing by using an ERP system.

A project was realized between this company and our research group to develop a model to organize, to present the information and to do savoir-faire ERP, which is systematically elaborated in the integration and deployment phase of an ERP system by customer companies. In addition, this model should allow the company to analyze the needs of current and future customers, with a goal to figure out the technological evolution of the company.

This project led to a master work that was supervised by our research group [37]. The model developed in the project was built using the SEAM method [4]. The SeamCAD tool is also used to build some part of this model that can be expressed by the SeamCAD modeling language. Even though the whole paper-based model built in this project cannot be expressed in SeamCAD, the following benefits are noticeable at the company for the part that is built in SeamCAD

- Model elements are explicitly represented using the formally-defined building blocks of the SeamCAD modeling language. Thanks to the rigorousness of the SeamCAD modeling language, the created model is well-formed
- It is particularly easier to browse the model created in the SeamCAD tool rather than turning the piece of papers in the paper-based version of the model to look for the right diagram. For instance, the value networks can easily be toggled between the whole and the composite. Diagrams can be scoped to show only the value network of interest.
- The hierarchical approach is powerful for representing business strategies

Figure 41 a) shows the very first organizational level of the model. At this level, there are two value networks on the market of manufacturing parts for watches: the supplier and the adopter. The supplier value network proposes products and services that would be consumed by the adopter value network.

Figure 41 b) shows the second organizational level of the model. At this level, the adopter value network is seen as composite. It has the customer, the distributor, the provider and the company that manages the ERP.

Figure 42 shows this company as composite. It has direct and indirect purchases, logistics, manufacturing, sale and marketing, R&D, financial infrastructure, infrastructure for making decision and COM which is the ERP all as working objects. There are two distributed actions (`sale_mfg of product` and `support of sale_mfg`) within the main distributed action of the company. The COM takes part in both. Note that properties that represent the products, objectives, results… in these working objects have the stereotype `<<in>>` or `<<out>>`. They are input and output properties that are exchanged within the company.

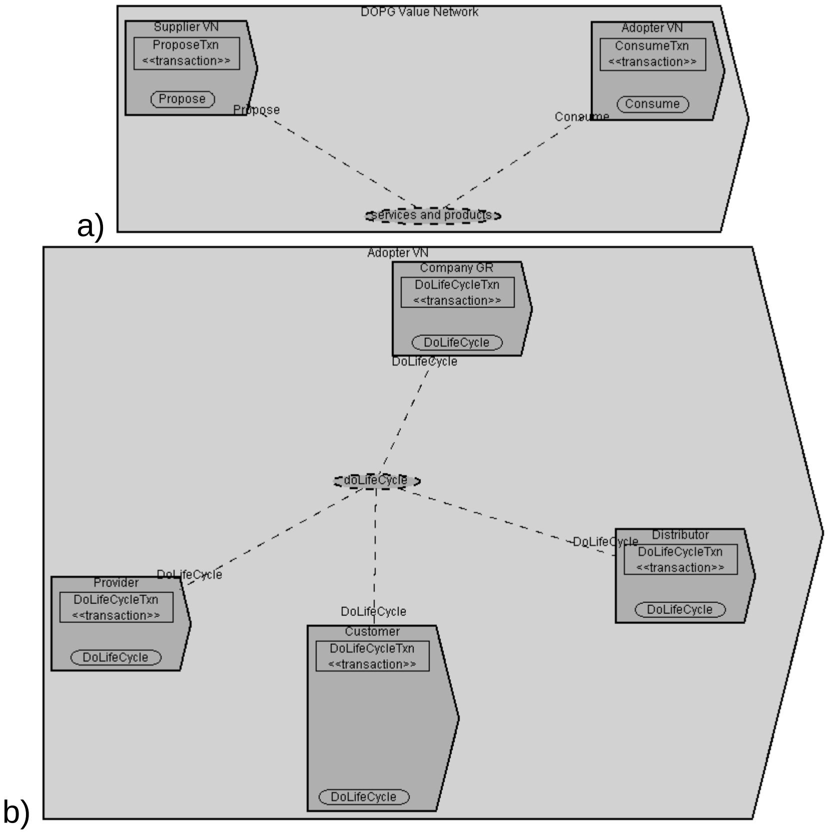


Figure 41. The supplier and the adopter value network in the market of watch parts manufacturing

Figure 43 shows the data managed by the ERP, which manages various kinds of data like items, sales, purchases, stocks, etc. These management activities are represented as distributed actions. In Figure 44, one of these distributed actions is scoped and zoomed in: `manage the sales`. This distributed action is broken down into three component distributed actions (listed in the order they happen): `handle the command`, `handle the delivery` and `handle the bill`. Again, the input and output properties of the working objects that participate in these distributed actions are stereotyped `<<in>>` and `<<out>>`, respectively.

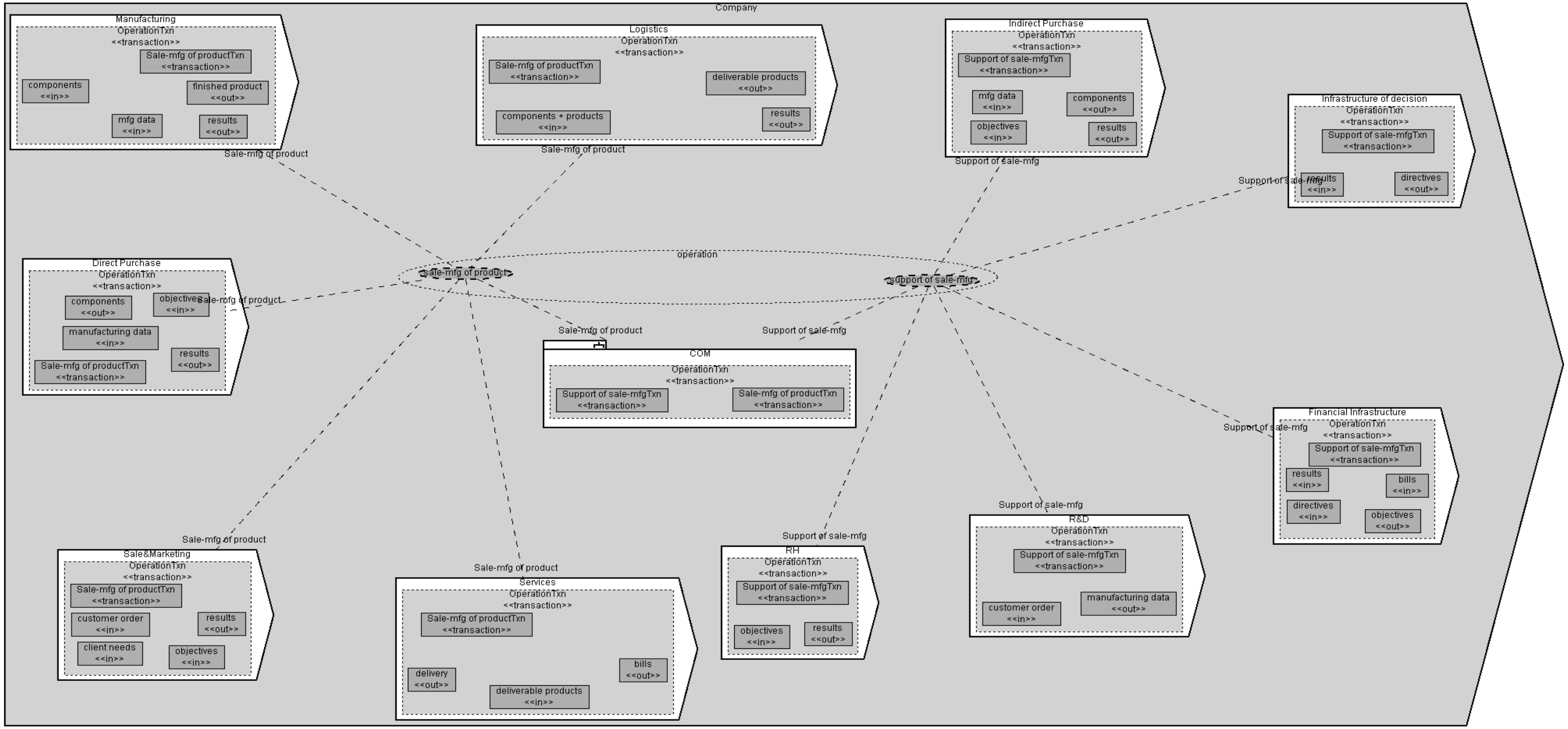


Figure 42. A third organizational level showing the ERP-seeking company

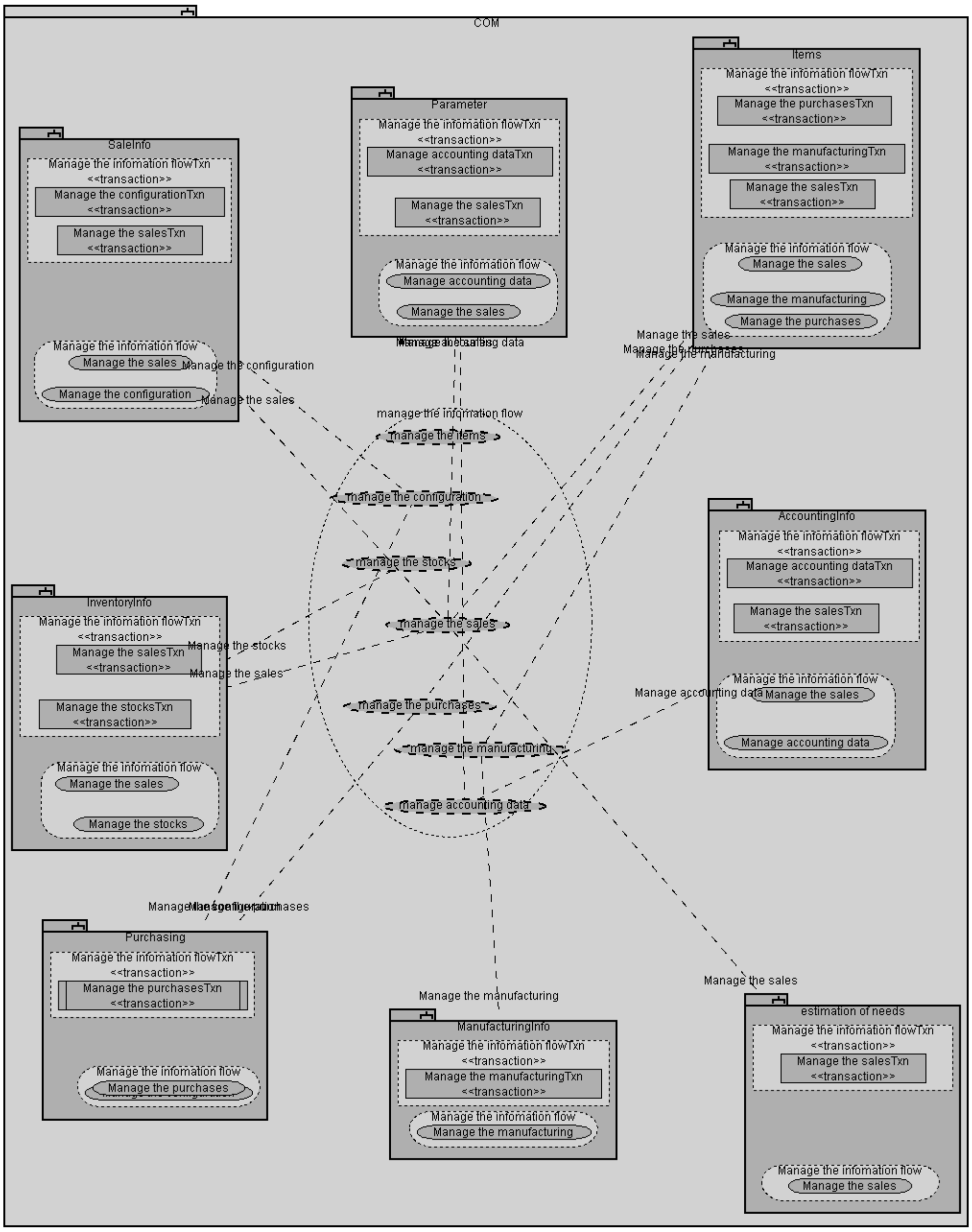


Figure 43. A fourth organizational level showing the ERP system

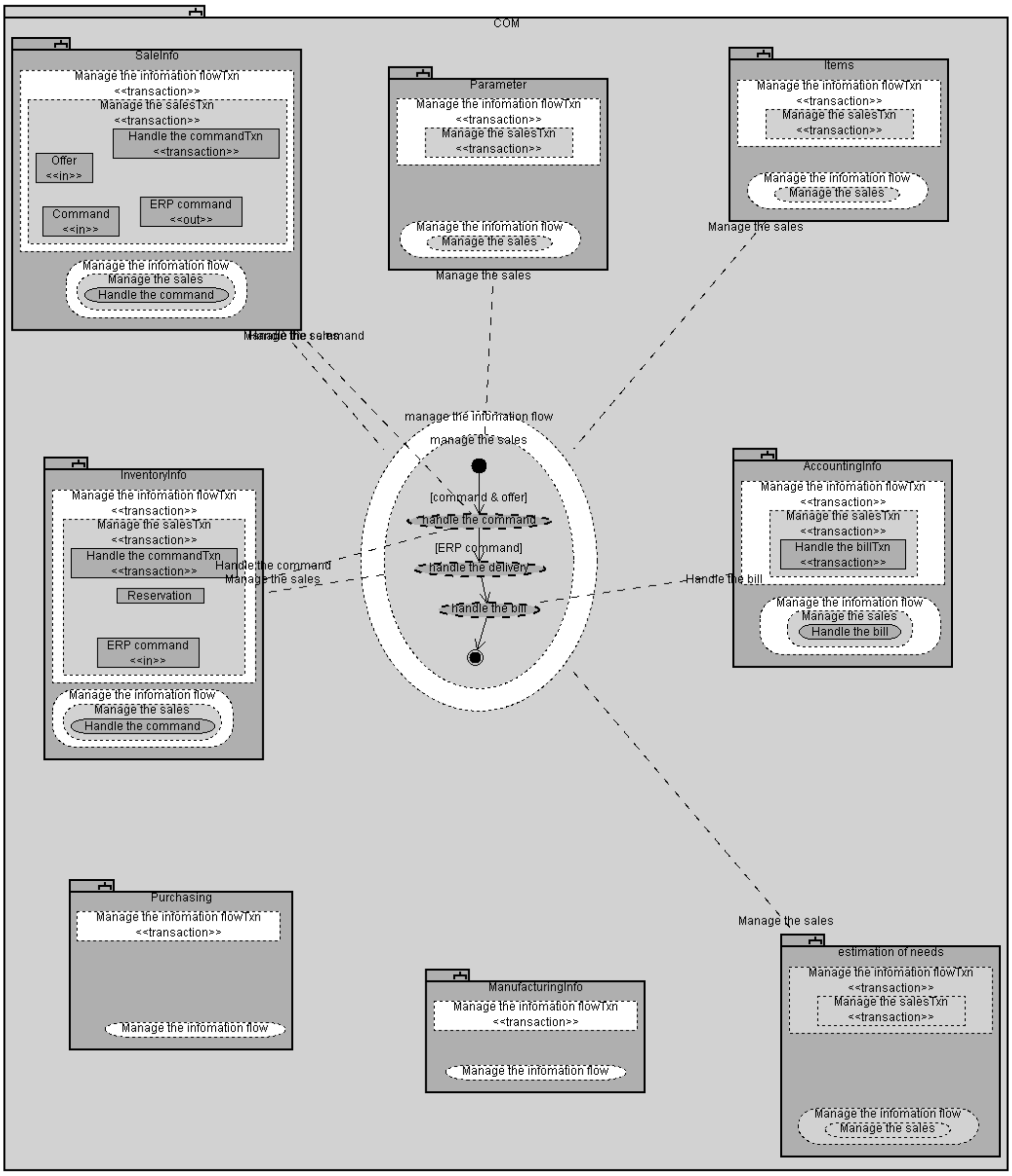


Figure 44. A fourth organizational level and a higher functional level showing data exchanged and sequences of component actions for the distributed action `manage the sales`

Although the enterprise model created in SeamCAD has the aforementioned advantages, the following drawbacks were noticed at the company where this project was carried out

- The value network, the company or the ERP system seen as whole were not very interesting
- In SeamCAD, localized actions are drawn as rounded rectangles, distributed actions as ellipses. The company did not really like this notation.
- Goal-belief modeling [38] is useful for describing the needs of customers but it is not supported by SeamCAD

5.1.3. Designing EA with the SEAM method and SeamCAD

A project was launched to apply the SEAM method and the SeamCAD computer-aided tool in designing an enterprise model for a project of a new building in our university. This project led to a master work that was supervised by our research group [39]. The model built in this project was useful to specify how the building should be equipped and what IT system should be installed in the building. The model was built using in the very first version of SeamCAD has proved its usefulness of showing different views that would be of interest of different partners in the project. The diagram shown in Figure 45 depicts the big picture relating to how the school `IAndC` conducts research and learning. The school is responsible for providing students, researchers with quality research and learning facilities. The industry may get involved in some research. The university management supervises this collaboration. This diagram represents the very first organizational level of the enterprise model.

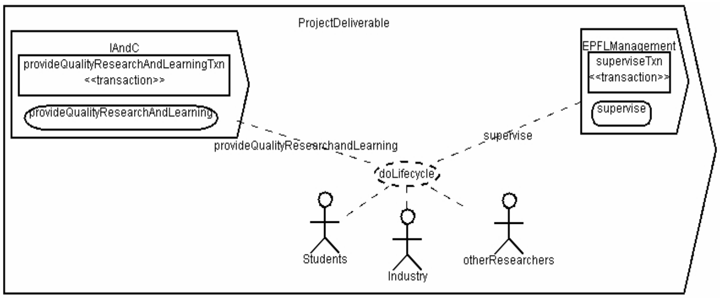


Figure 45. The 1st organizational level shows the management of the university and the school of which the new building was constructed.

Figure 46 a) depicts the internal structure of the school `IAndC`. In this school, the business support system `BSS` collaborates with professors, researchers, teachers, students and visitors. Figure 46 b) details the collaboration between `BSS` and them. Note that the two views given by Figure 46 are at the second organizational level of the enterprise model.

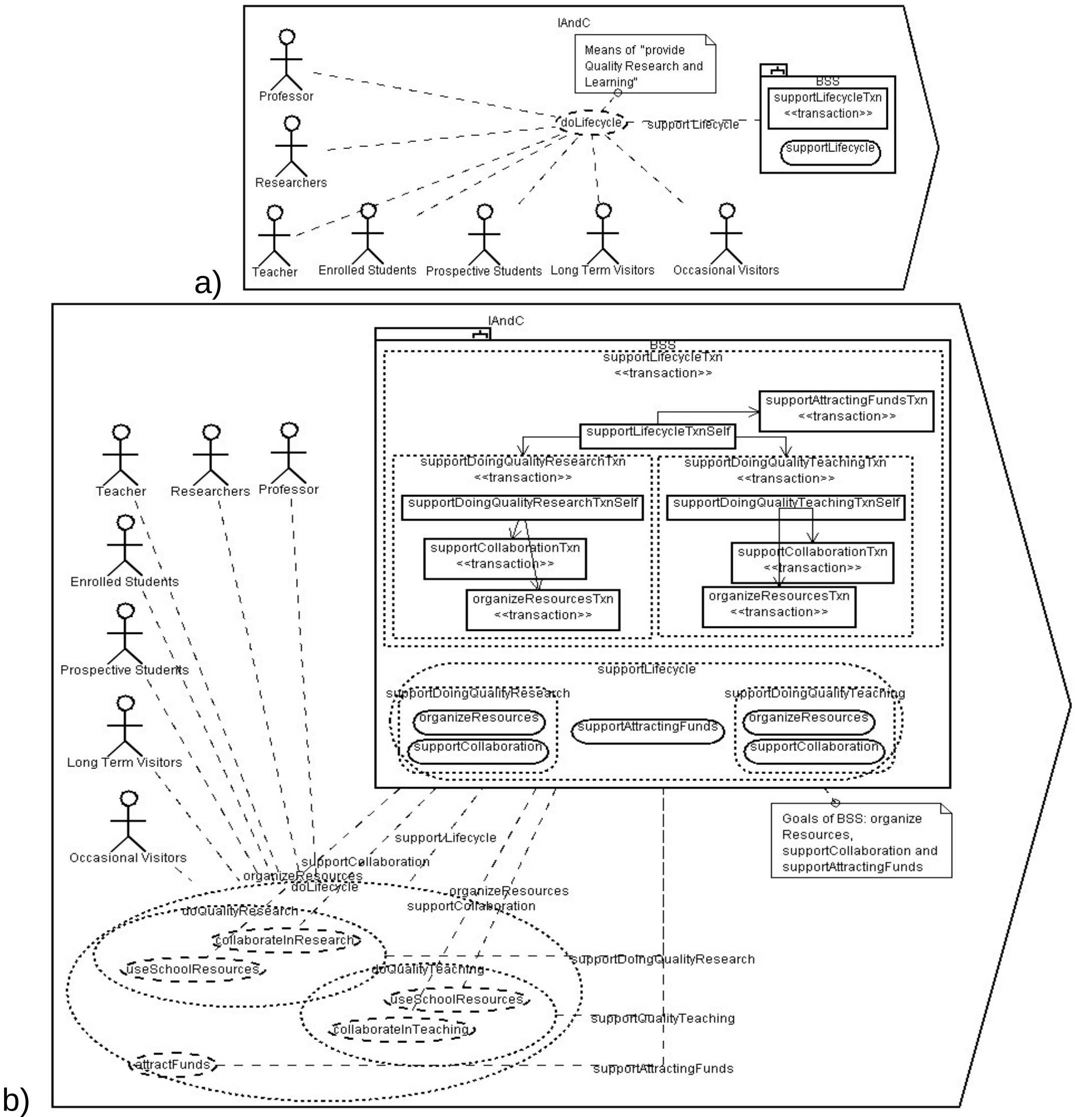


Figure 46. The 2nd organizational level shows
the internal structure and business of the school `IAndC`.

Going deeper into the business support system `BSS`, Figure 47 diagrammatically describes modern channels of the new building of the school `IAndC` and how they are exploited. An IT application is developed to help different departments of the school efficiently exploit these channels: accessing the web, accessing school events, reserving a room, etc. The diagrams shown in Figure 47 represent the third organizational level of the enterprise model.

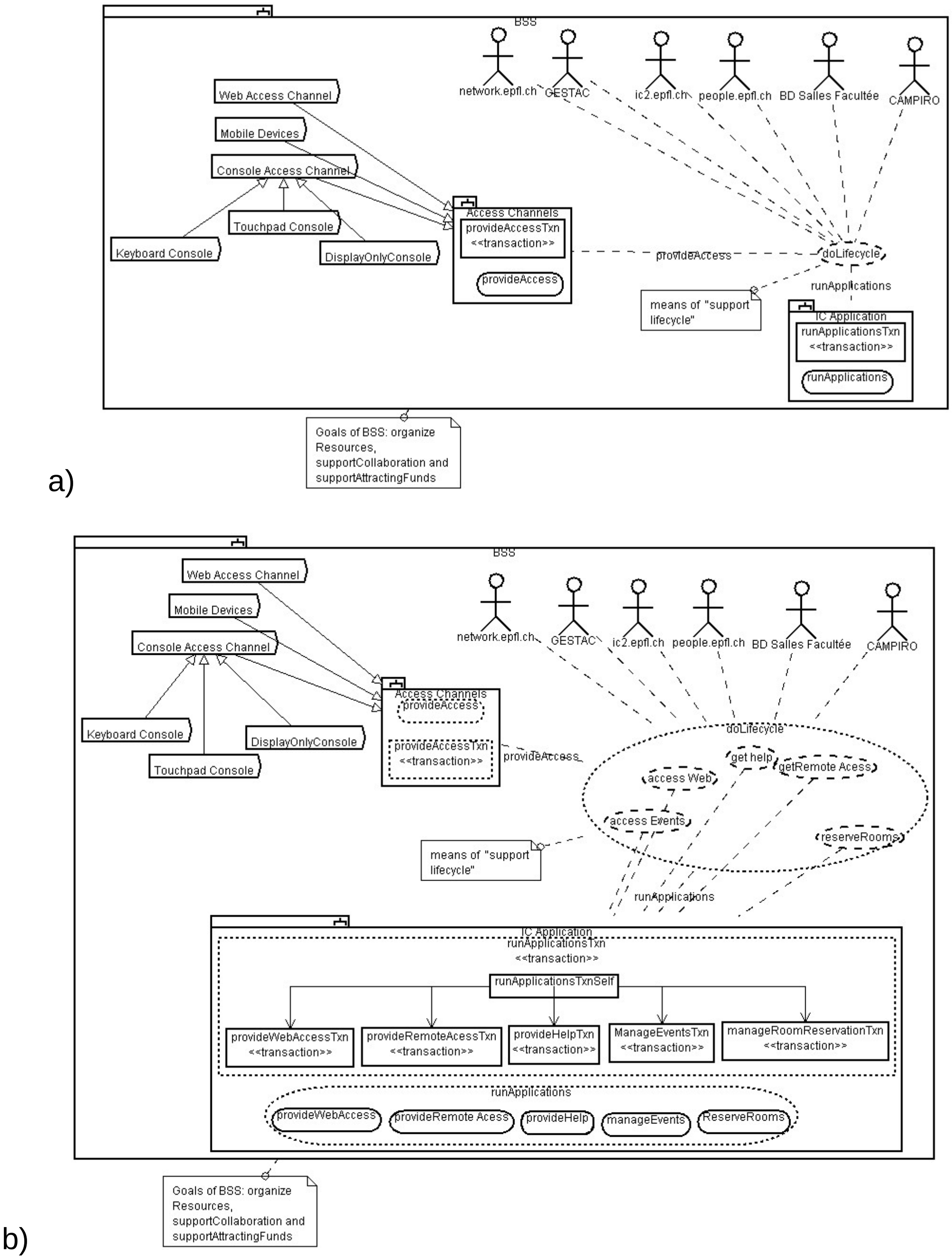


Figure 47. The 3rd organizational level shows modern channels equipped by BSS and an IT application through which people can efficiently exploit these channels

The lowest organizational levels are addressed in Figure 48. The diagram shown in this figure is dedicated to the component structure of the application that helps people access channels equipped in the new building of the school IAndC.

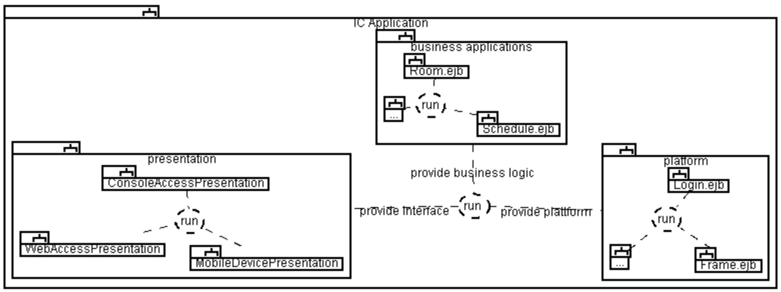


Figure 48. 3rd organizational level shows equipments managed by `BSS` and an IT application through which people can effectively exploit the equipments

Throughout this master project, advantages, shortcomings and some research issues of the SEAM method and the SeamCAD framework were discussed. Most notably, it was evident that

- Making the hierarchy explicit was very good point in the project
- It was possible to represent multiple units of interest in the model
- It was possible to maintain the traceability between different views of the model edited the SeamCAD tool
- The fact that system boundaries are always respected makes the enterprise model in the SEAM modeling language rigorous but inflexible.
- Diagrams in SeamCAD tend to get huge, making it hard to further elaborate if the model has a large number of objects and actions.
- It is not yet clear whether the model elements created in the SeamCAD are types or instances.

5.1.4. Simulation of System Dynamics with SeamCAD

A semester project was carried out under the supervision of our research lab with a goal to define an extension to the SeamCAD modeling language and to implement a separate tool to be able to quantitatively analyze a business by simulating a specific action or behavior during a predefined time period with SeamCAD [40]. This language extension is based on System Dynamics – a methodology for studying and managing complex feedback systems [41]. As its name indicates, this methodology proposes a way to look at dynamic behavior of systems in terms of changing patterns over time.

The most important concept of System Dynamics is the use of feedback thinking [41]. In System Dynamics (SD), the feedback has a loop structure. Let’s consider an imaginary arms race between the U.S.S.R. and the U.S. [42] as illustrated in Figure 49. Each nation builds its arms stockpile based on the threat imposed by the other, which is directly determined by its arms stockpile. It is represented as a looped feedback. The + sign implies that this feedback is positive. If the stockpile is represented as a mathematical variable, its value grows exponentially over time as illustrated in the graph to the right of Figure 49.

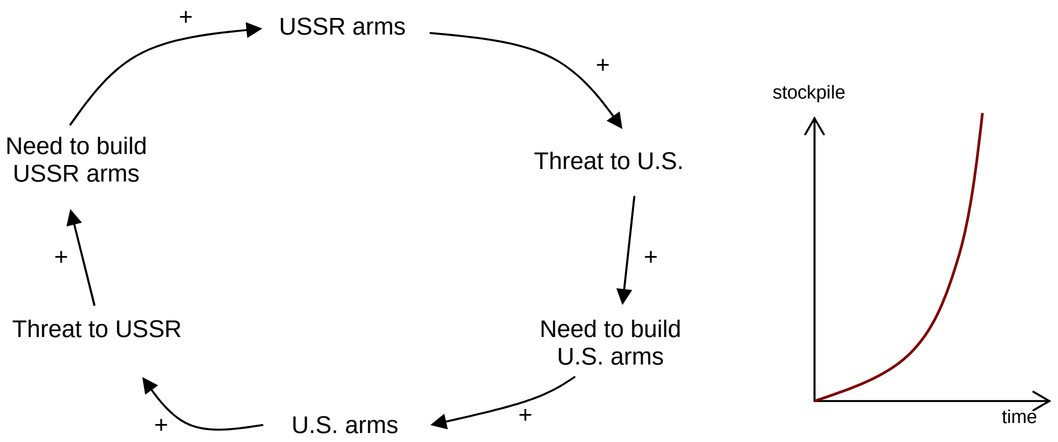


Figure 49. An example of positive feedback in System Dynamics

A SD model consists of three types of entities, namely Stocks, Flows and Information objects. In SeamCAD, model elements are classified by means of stereotypes into the following categories

- Level: corresponds to a stock of material accumulation in SD. It is defined by an initial value and a differential equation expressing its evolution in time.
- Rate: stands for a flow that feeds a level or depletes it. It can be defined by a mathematical equation evaluated at each step in the simulation time.
- Parameter: counterpart of information in SD. Can be defined by a mathematical equation. A parameter can influence a level only through a rate.
- Constant: counterpart of information in SD. It is defined by an initial value that is unchanged throughout the course of the simulation.

To be able to capture represent these quantitative values in SeamCAD, notes containing SD model are attached to model elements. A math-like sub language for what is written in these notes is defined. This sub language includes mathematical expressions, simulation options and model requisites.

The implementation of this project resulted in a plug-in application that can be invoked from the SeamCAD tool to simulate the currently-edited modeling that follows the SeamCAD-extended language. This plug-in takes the XML data generated by the SeamCAD tool for model being edited and extracts the notes that contains data necessary for simulation. It finally renders a graphical simulation over time based on quantitative parameters and values described in the model edited in the SeamCAD tool.

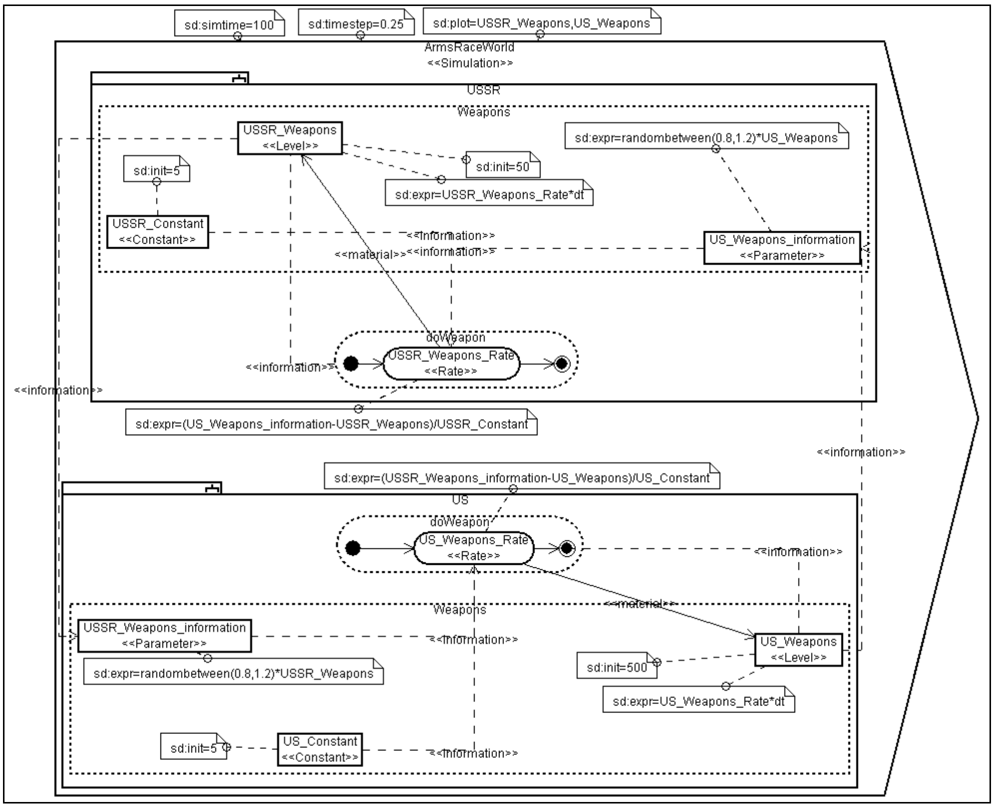


Figure 50. Arm’s Rate model made in SeamCAD using SeamCAD-extended language

Figure 50 gives a diagram of a model created in the SeamCAD-extended language to describe the arms race example. This model exemplifies the typical goal seeking and the exponential growth behavior of a SD model. The USSR and the U.S. are represented by two working objects. The working object `U.S.` has localized action named `US_Weapons_Rate` that represents the arms-building action of the U.S. The stockpile of this nation is represented as a property named `US_Weapons`. There are also two other properties (`US_Constant` and `US_Weapons_Information`) that represent the SD constant and information. A similar representation is made for the U.S.S.R. Figure 51 shows an output of the simulation for this model using the plug-in application implemented in this project. In this figure, the red line represents the weapon stock of the U.S.S.R. over time and the blue line stands for that of the U.S.

The results of this project include an extension of the SeamCAD modeling language and the implementation of a graph-drawing plug-in application that can simulate the SD model created using the SeamCAD-extended language. These results were tested for the arms race example. More work should be done to generalize these results to deal with more sophisticated SD models.

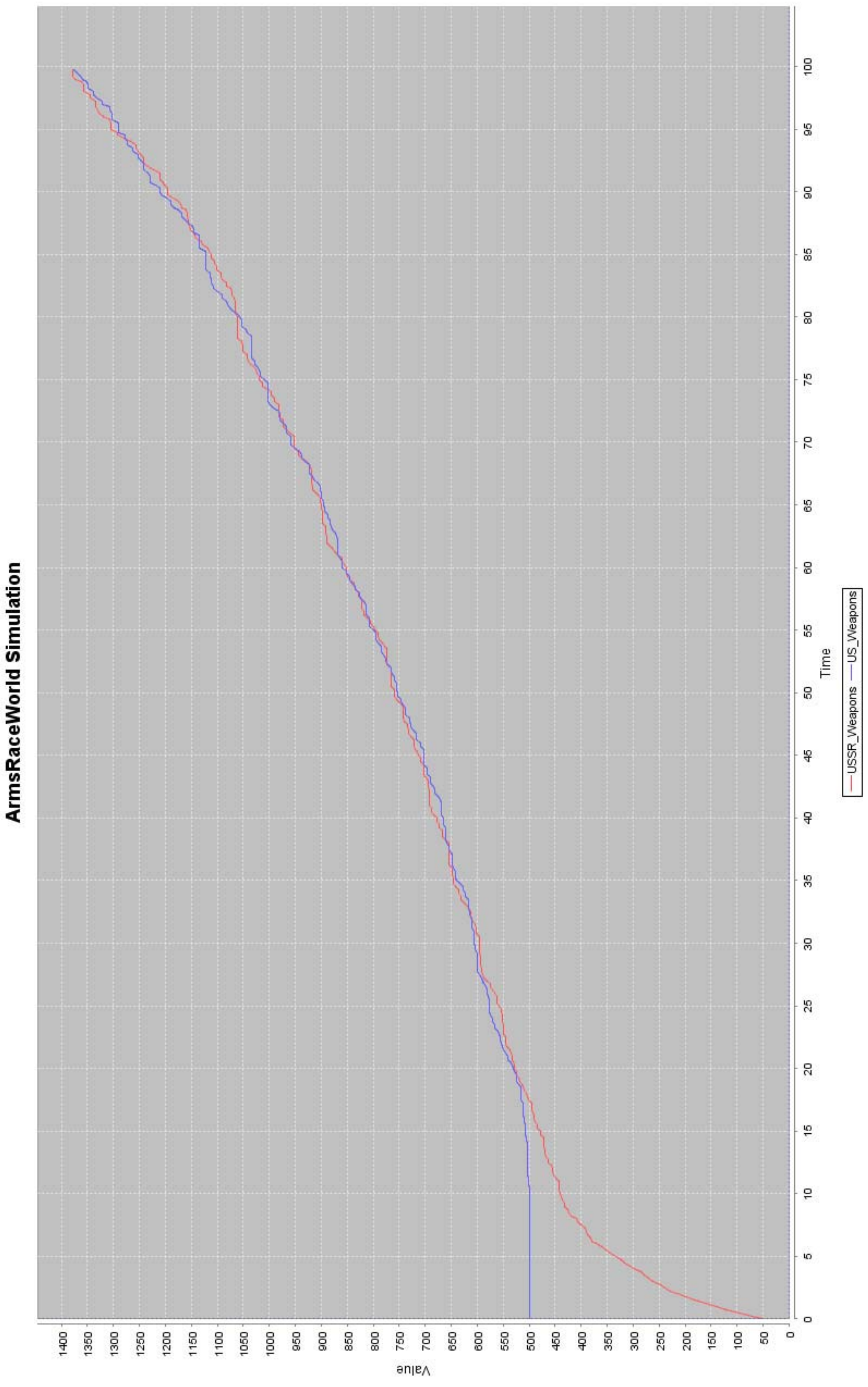


Figure 51. Simulation output of the Arm's Rate model using the simulation plug-in.

5.1.5. Lessons learnt from building enterprise models in SeamCAD

After having worked with SeamCAD in making enterprise models, the people in the companies and the master's students (we call them the modelers) who got involved in the projects presented in subsections 5.1.1, 5.1.2 and 5.1.3 expressed their opinions and show some reactions. The following 4 points summarize what they learnt from building their enterprise models in SeamCAD

- Although the modelers could get familiar with the modeling terms of SEAM [4] before actually using SeamCAD, the modelers were able to understand the notion of hierarchy and whole/composite efficiently by using the SeamCAD tool to build their enterprise model.
- The modelers learnt how to browse their enterprise model with the SeamCAD tool by interacting with an editing window that focuses on certain part of the model they want to view.
- The model that was created using the SeamCAD tool can be considered as an electronic version of the enterprise model as opposed to the hard version that was built using pieces of paper and a pencil. The paper-based version of the enterprise model was typically built using some pre-built template diagrams (e.g. templates for the market level, the value network level, the company level and the IT level). In the electronic version, the modelers could open more diagrams than they did in the hard version. More concretely, diagrams of the enterprise model can be generated in the SeamCAD tool by just choosing the context working object and the functional level within this working object. Customizing the diagrams created using pre-built templates in the hard version was technically possible, but it required a lot of drawing burden on pieces of paper.
- The modelers were able to understand the notion of model well-formedness that is defined in the SeamCAD modeling language. The SeamCAD tool prevents them from adding model elements to their enterprise model in a way that violates the well-formedness rules defined for the SeamCAD modeling language (e.g. the tool does not allow them to create a property inside a distributed action).

5.2. Feedback from Practitioners, Researchers and Students

To see how SeamCAD addresses the four modeling challenges (uniformness, multi-entity, granularity and well-formnedness) that were identified in Chapter 1 and if it brings some value to the industry and university courses, we should validate it. During the last months of this Ph.D. work, practitioners or researchers in different domains related to Enterprise Architecture and students who took our master's course in Enterprise Architecture were invited to participate in the validation of SeamCAD.

5.2.1. Protocol for getting users' feedback

A questionnaire was prepared to get feedback for SeamCAD from practitioners/researchers in EA-related fields. This feedback should be about how they evaluate to which extent SeamCAD addresses the four modeling challenges that are presented in Chapter 1. It was then extended to cover additional interesting issues like top-down versus bottom-up modeling, the intuitiveness of notation and the right term to address the SeamCAD tool as a modeling tool. To fill in this questionnaire, the practitioners obviously need to get acquainted to SeamCAD beforehand. Therefore, a scenario was written to provide them with step-by-step instructions of how to practice the SeamCAD tool. The scenario and the questionnaire were tested within our research group to estimate the time needed for each participant to complete the validation and to check if they had anything unclear.

It was very hard to get the practitioners/researchers that were willing to participate in the validation all at the same time due to the difference of their availability. We decided to work with each participant individually. In case no face-to-face conversation could be established, the participant can follow the scenario and then fill in the questionnaire himself while getting assistance either on the phone or an online chat.

For the students who were willing to take part in the validation, we defined another protocol. After finishing their exams at the end of the semester, they could have time to work with us for several hours or even a whole day. It was thus possible to get them altogether or by a group of three to five at the same time. Instead of following a predefined scenario, the students should be able to do more extensive practice by building an enterprise model in the SeamCAD tool. The questionnaire used for obtaining feedback from the practitioners should be slightly modified to reflect what the students would do.

5.2.2. Ratings

A total of 11 practitioners/researchers participated in this validation. They followed a 2-phase tutorial with some assistance (with minimal influence on their viewpoints) of the person responsible for the tool. In the first phase, each practitioner tried to view an existing model of the online bookstore model to get acquainted to basic functionalities of SeamCAD tool and the fundamental modeling concepts SeamCAD modeling language. She/he then proceeded in the second phase to extend one more organizational level and two functional levels of the model of the bookstore. The tutorial is fully presented in the Appendix B. After having practiced the tool, each practitioner was asked to answer a questionnaire to rate how SeamCAD meets the four modeling challenges and to give their suggestions. Each practitioner/researcher completed the tutorial and the questionnaire in about 30-45 minutes.

A total of 9 students took part in the validation of SeamCAD. They had just finished a master’s course on EA and Service Oriented Architecture given by our lab. In this course, students were asked to make an enterprise model using Service-Oriented Architecture for imaginary company – a company that manufacture and sell lightweight aircraft diesel-powered engines. They were divided into groups of less than six. Each group represented a company and built an EA model for their company in paper. Students who participated in the validation of SeamCAD were asked to rebuild the model they made in paper using the SeamCAD tool. They then answered a questionnaire to evaluate how SeamCAD meets the four modeling challenges and also to give their feedback as if SeamCAD was used as a teaching tool in the course. It took approximately 7 hours for these students to complete the validation.

20 participants gave their answers and feedback. The two questionnaires used in this validation (one for practitioners, one for students) have 7 questions in common. Table 14 gives the distribution of their answers for these questions.

Table 14. Distribution of answers given by practitioners and students who practiced SeamCAD

Question	Answer distribution of 20 participants			
	Excellent (Yes)	Good	Bad	Very bad (No)
Do you think that the top-down approach of SeamCAD has some value in your practice?	20/20	--	--	0/20
How do you rate the way SeamCAD manages organizational levels?	7/20	13/20	0/20	0/20
How do you rate the way SeamCAD manages functional levels?	4/20	15/20	1/20	0/20
How do you rate the way SeamCAD handles multiple system representation?	10/20	8/20	2/20	0/20
How do you rate the feature of SeamCAD through which diagrams can be opened as partial views of a common model?	13/20	7/20	0/20	0/20
How do you rate the way diagrams can be customized by hiding / showing specific elements in SeamCAD?	9/20	7/20	4/20	0/20
How do you rate the intuitiveness of the notation scheme used in SeamCAD?	4/20	14/20	2/20	0/20

As we can see in Table 14, all 20 participants, be practitioners, researchers or students, were satisfied with the way SeamCAD manages the organizational hierarchy (more than half of them rated this feature excellent). They also agree that the top-down approach brings value in their work. However, their opinions vary over the way SeamCAD manages the functional hierarchy. The majority of them rate it as good whilst the minority of them considered it as excellent. In particular, one of them rated this feature as bad. For the capability to represent multiple business entities or IT systems in a single enterprise model, half of the participants chose the excellent rating. Two of them chose the bad score whiles the others rated this feature as good.

All participants like the way SeamCAD manages a coherent model and generates a diagram as partial views of the model (more than 50% ratings are excellent). However, they were not really convinced by the feature that allows them to customize their diagrams by hiding/showing specific working object or distributed action (bad ratings are one fifth of the answers). Most of the participants found that the SeamCAD notation was intuitive, but two of them did not think so.

The distribution of answers given presented in Table 14 indicates that SeamCAD manages the organizational hierarchy in a better way than it does for functional hierarchy. The users appreciated the coherence of enterprise models edited in the SeamCAD tool more than the possibility to customize diagrams rendered in the tool by hiding (and showing) model elements. The notation schema is fairly good but is still not intuitive to some users. More work should be done to improve the functional level modeling, the diagram customization and the notation scheme.

5.2.3. Suggestions from the practitioners

Some features of SeamCAD were appreciated by the practitioners. The feedback of practitioners the following features were most liked

- The ability to open multiple windows at the same time that show different part of the enterprise model. These windows are kept in synch by the tool.

- The explicit organizational hierarchy of the enterprise model is visible in every window of the tool
- An overview of the whole enterprise model is visible in every window of the tool
- The context working object in a diagram makes the context view explicit especially when modeling an IT system
- The way the entire enterprise model is browsed by changing the organizational levels and/or the context working object shown in the diagram of an editing window of the tool
- The clear separation between the whole and the composite of a model element in the tool and the ability of toggling it between the whole and the composite in a diagram with ease
- The possibility for abstracting a property, a distributed action or a localized action and showing it in details at a higher functional level when needed.

However, the practitioners also left their suggestions in the questionnaire. In SeamCAD, a distributed action and all working objects that participate in it are supposed to be at the same organizational level. It was suggested to represent the collaboration between working objects at different organizational levels.

In the SeamCAD tool, for the sake of simplicity, all kinds of relations are represented by the same pictogram in the toolbar. The tool determines the specific relation kind (e.g. transition, association, participation) as soon as the user commits the creation of the relation based on the two model element that it connects. One practitioner got confused by this feature. The tool would be closer to the UML community if all kinds of relation are explicitly visible in the toolbar.

The tool needs some advanced features for navigating in the model. If the modeler frequently changes the context working object, the organizational level or the functional level, she should be allowed to go back and forth between these levels just like the way people surf the internet on their browser. In addition, as more and more model elements are created in the model, a search function that allows the modeler to easily grab the working object she is most interested in becomes apparently necessary.

Some suggestions address the semantics of action. In SeamCAD, it is possible to enter textual pre-condition and post-condition for a specific distributed action or a localized action. It was suggested that this textual description should be bound to properties which are changed by the action described.

The way SeamCAD imposes the name of a role played by a working object was sometimes confusing to some practitioners. It was suggested to develop a better naming convention for the role name rather than taking the name of the corresponding localized action.

For user-interface issues, it was suggested to visually represent hidden working objects or distributed actions under a special symbol. Then the hidden elements can quickly be shown by clicking on this symbol. Another suggestion encourages the use of pop-up widgets to provide the modeler with more information on model elements of her interest.

At the end of the questionnaire used for getting feedback of the practitioners, there was a question about how the practitioners could classify the tool SeamCAD. Possible terms were: Computer-Aided Enterprise Modeling tool, Computer-Aided Requirement Engineering tool, Computer-Aided Design tool or a new term. All participants picked the first answer agreeing that the SeamCAD tool could be regarded as a Computer-Aided Enterprise Modeling tool.

5.2.4. Suggestions from the master's students

The students that participated in the validation were kindly to leaves their detailed comments while practicing the tool as if it was the teaching tool in the master's course they had taken. In overall, most of them felt that the SeamCAD tool could be used as a teaching tool as they found it was generally interesting to model their imaginary company using a computer-aided tool after having made a paper-based model. The tool allows them to browse their enterprise models in a more efficient way than they did on pieces of paper. In addition, after using the tool, they understood the notion of hierarchy in their enterprise model more clearly. However, they had some difficulty in using the tool and they made some suggestions for improving it.

The following problems was noted during the validation

- It was difficult to correctly understand the level of granularity. It is not intuitive to toggle a distributed action to be able to add component localized actions and component properties in working objects that participate in it.
- Quite a lot of assistances were needed before the students could get familiar with the tool
- The tool was sometimes buggy. In some cases, elements that were wrongly created could not be easily corrected. They had to be deleted and re-created.
- It was quite hard to master how to create a line to represent a relation using mouse-click
- Under the Java look and feel on Mac, the user-interface does fully function. This problem could be down to the implementation of the Java virtual machine on Mac.
- It was difficult to understand the how the properties and the localized actions can properly be used. It is also difficult to imagine the input and output parameter solely based on their stereotypes.
- The way the tool resizes enclosing pictograms by moving their nested pictograms around was not intuitive.
- Icons representing different element kinds in the toolbar do not really look different. The ellipse, the rounded rectangle and the regular rectangle somehow look the same.
- Diagrams tend to get very wide. The automatic layout function did not efficiently work.

Having experienced the aforementioned problems in practicing the SeamCAD tool, the students made the following suggestions

- Enclosing pictograms should be manually resizable. It would be more convenient to drag and drop pictograms.
- Notation used in SeamCAD and that in the course, especially for relations, could be unified
- A complete user manual for SeamCAD would help students a lot
- Zooming diagrams is necessary to see the link between all diagrams opened in the tool. If no more than one modeler can work concurrently on a model, making paper-based diagrams could be the better choice.
- Relations should be created with ease like many graphical editors. Clipboard operations for model elements are in needs.

5.2.5. What was learnt from the feedback?

The feedback for SeamCAD left by 20 practitioners and students suggest that the modeling language and the tool have some advantages as well as disadvantages. Most notably, SeamCAD is generally good for representing the organizational hierarchy, the context view of an enterprise model in a hierarchical way. Separating the whole and the composite with the possibility to toggle any model element between these two views can be considered as good points too. Managing the well-formedness of the enterprise model and the browsing capability are also noticeable advantages, especially of the SeamCAD tool. This tool might safely be used as a teaching tool in course on SEAM and Enterprise Architecture.

Shortcomings of SeamCAD include the way the functional hierarchy is represented in SeamCAD, the lack of advanced features for managing large enterprise models and in customizing diagrams, the non-intuitive way of expressing information flow as the exchange of input and output properties, the instability of the tool due to bugs. They should be taken into account for improving SeamCAD.

Conclusion and Future Work

In Enterprise architecture (EA), the goal is to align the business resources and IT resources in order to improve the enterprise competitiveness, for example, by gaining more customers, reducing the operation costs and complexity or responding to changes with agility. An enterprise model that represents the enterprise and its environment may include various aspects such as the internal structure of enterprise and the services provided by the enterprise, the business processes and data flow between business entities, the IT components and their interaction. Given the fact that people often simplify their perception of the reality by analyzing it hierarchically, we decided to develop a hierarchy-oriented framework for modeling the organization and the services of the enterprise. Developing such a framework has four challenges. First, a systematic modeling approach should uniformly be applied to model all business entities and IT systems. Second, multiple business entities and IT systems can be represented in detail, for instance as a black-box and as a white-box. Third, the interaction between business entities and IT systems, as well as their behavior, can be represented at different levels of granularity. Four, the enterprise model should be coherent and the consistency between different views of the same enterprise model should be maintained.

SeamCAD – the main contribution of this dissertation is such a modeling framework. It is part of the method SEAM developed in our research group. SeamCAD consists of a modeling language and a computer-aided tool, which together consolidate the SEAM method by defining the hierarchy levels, modeling building blocks, the well-formedness rules, the notation and by formalizing them in a declarative language Alloy and realizing them in the computer-aided tool implemented in Java.

In the SeamCAD modeling language, two kinds of hierarchy are defined: organizational hierarchy and functional hierarchy. The organizational hierarchy describes the organization of the enterprise being modeled and of its environment. This hierarchy is formed by a series of organizational levels. Basically, the business entities and IT systems, or components of the enterprise being modeled, are organized into organizational levels. The functional hierarchy is composed of a series of functional levels each of which captures the service or the interaction at a different level of granularity. To model the business entities and IT systems or components of the enterprise, we define the building block *working object*. To model the interaction between working objects, we define the building block *distributed action*. To represent the externally-observable properties and services of a working object, we define the building blocks *property* and *localized action*, respectively. These four building blocks originate from the two basic modeling concepts of RM-ODP: object and action (the working object and the property are two different kinds of ODP object whereas the distributed action and the localized action are two different kinds of ODP action).

In SeamCAD, any model element can be seen either *as whole* or *as composite*. These two ways of viewing a modeling element are based on the atomicity and the composite – two interpretation concepts of RM-ODP. A working object seen as whole is characterized by its externally-observable properties and localized actions. A working object seen as composite consists of component working objects and possibly the distributed actions in which they participate. A property seen as whole exhibits attributes like name and stereotype. A property seen as composite has component properties in addition to its attributes. The component properties can be put in relation by UML-like *association* or *generalization*. A distributed action seen as whole exhibits attributes such as name and stereotype. A distributed action seen as composite has component distributed actions in addition to its attributes. It is possible to specify the order between component actions by means of a UML-like *transition*. Therefore, a distributed action seen as composite can be regarded as a UML activity. Similarly, a localized action seen as whole exhibits attributes such as name and stereotype. A localized action seen as composite has component localized actions in addition to its attributes. The order between this component localized actions can be described by means of a UML-like *transition*, making a localized action seen as composite look like a UML activity. There is also the relation *participation* that relates a working object to a distributed action in which it participates.

The semantics of the enterprise model built in the SeamCAD modeling language is defined not only in terms of model elements instantiated from the four aforementioned building blocks but also by the relations between these elements. There are two kinds of relations in SeamCAD: intrinsic relations and diagrammatically-presented relations. The former includes the *containment* and the *binding*. The latter are aforementioned UML-like relations. Containment is the relation between a model element and its component elements. There are two forms of binding that correspond to two kinds of action in SeamCAD. *Goal binding* is the relation from a distributed action to the properties and the localized actions of all working objects that take part in it. *Means binding* is the relation between a localized action and the distributed action that implements it. The intrinsic relations may not diagrammatically be rendered in any diagram of the enterprise model. They are necessary to maintain the coherence of the enterprise model so that diagrams can be generated as partial views of the model.

The following UML-like relations are defined in SeamCAD: association (between properties), generalization (between elements of the same kind), action transition (between distributed actions or localized actions) and participation link (between a working object and a distributed action). These UML-like relations can be presented diagrammatically under lines with various drawing patterns.

The SeamCAD modeling language was not only informally defined in terms of verbal description of building blocks and modeling terms but also rigorously defined by means of a meta-model and its formalization. The meta-model consists of a UML class diagram that express all building blocks and a total of 19 well-formedness rules that govern the manner model elements instantiated from the building blocks are put together in an enterprise model. The meta-model is formalized in Alloy – a declarative modeling language based on first order logic and set theory.

The SeamCAD computer-aided tool was specifically developed for the SeamCAD modeling language. The tool manages a coherent enterprise model and allows the modeler to edit or view the model at any organizational level and functional level of her interest. The entire model can be viewed and browsed by means of a tree-like navigation panel. Diagrams can be generated for any organizational level and functional level. The notation of SeamCAD mimics that of UML but with two main changes. First, a block arrow pictogram is introduced to express business entities as a working object. Second, most pictograms can be nested to visually show component elements of the model element being represented as composite.

The SeamCAD modeling language and tool were evaluated by practitioners and students. A total of 11 practitioners were invited to test the SeamCAD tool for about 45 minutes. 9 master's students in our university participated in a one-day session in which they remade an enterprise model that they had previously made on paper during a master's course given at our university. All 20 participants, be practitioners and students, were asked to fill in a questionnaire to rate the way SeamCAD addresses the four modeling challenges and also to give their suggestions. Regarding the first challenge, all 20 participants were satisfied with the way SeamCAD manages the organizational hierarchy. However, their opinions vary over the way SeanCAD manages the functional hierarchy and the capability to represent multiple systems in a single enterprise model, which correspond to the second and the third challenge. For the fourth challenge, all participants like the way SeamCAD manages a coherent model and generates a diagram to show some partial representation of the model. The suggestions from practitioners and students point out some limitations and open some directions for improving the SeamCAD language and tool.

Several research directions are opened to further this Ph.D. work. The suggestions given by practitioners and students who used SeamCAD should be taken into account for improvements. For instance, the way the modeler browses her enterprise model can be more sophisticated with a back-and-forth navigation mechanism. Another possible improvement is to capture the information flow between working objects in a more visual way. Yet another possibility is to extend SeamCAD to include goal-belief modeling [38] or to merge the two modeling frameworks.

The semantics of the distributed action and the localized action in the SeamCAD modeling language can be enriched to enable the portability of the enterprise model edited in the SeamCAD tool. Most notably, we can define a grammar for the pre-conditions and the post-conditions for action. The pre-conditions and the post-conditions of a distributed action should be defined in terms of the properties (at the same functional level) of the working object that participates in the action. The pre-conditions and the post-conditions of a localized action should be defined in terms of the properties (at the same functional level) of the same working object. Alternatively, we can define a rule for naming a transition between two localized actions of the same working object, e.g. we can put the names of the properties that are produced or consumed by these actions on this transition to diagrammatically express their semantics (as illustrated on the reparation process of aircraft engines presented in Subsection 5.1.1). As such, the enterprise model created in SeamCAD can be exported either to another diagrammatic language such as BPMN or to some code (a declarative language such as Alloy as exemplified in more details in Appendix A or an execution language such as BPEL, provided that BPEL is extended to include human activities and services of non-IT entity [43]).

The link between the formalization of the SeamCAD modeling language and the implementation code that implements its computer-aided tool should be established to make the entire SeamCAD framework customizable. Apparently, the Java code that manipulates the data structure managed by the tool is strongly influenced by the Alloy code that formalizes the modeling language. Ideally, we can generate the imperative Java code from the declarative Alloy code but this approach is very hard, if not unrealistic, because it basically deals with one of the most difficult problems in software engineering: the gap between specification and realization, which by nature lies at two very different levels of abstraction. Considering that DSM (Domain Specific Modeling) can increase the abstraction level of the specification while keeping the realization translatable from the specification by narrowing down the specification language into a specific domain [44], it could be a good research direction to investigate how the SeamCAD modeling language can be formalized using a certain DSM language instead of Alloy so that the partial implementation code of SeamCAD tool can be mapped from it. In this approach, the SeamCAD modeling language may not freely be customized but the mapping from its formalization to the partial implementation code should be feasible.

Appendix A: An Example of Specifying the Semantics of Actions and Refinement Principles in SeamCAD

In this appendix, the semantics of actions and design principles in building enterprise model will be illustrated through an example of an online bookstore – the same example that is presented in Section 1.2 of Chapter 1. Different modeling approaches will be used for specifying the semantics of actions and how the actions can be refined. In each of the following subsections, diagrammatic representation and Alloy code that describe actions are given for an approach. Using Alloy Analyzer[15], the Alloy code can be checked for consistency and can be executed. The possibility to produce BPEL code out of this diagrammatic representation is also discussed.

First, the market, the bookstore value network and the customer are coded as Alloy signatures in the following code fragment. They are named using a simple naming convention: a postfix "_C" indicates that the working object is seen as composite while a postfix "_W" implies that the working object is seen as whole. The Bookstore Value Network seen as whole has the following properties: a book catalog, an inventory of book and cash. Note that the inventory and the cash may change overtime. They are declared as a mapping from a set of books and an integer number to the concept of time, respectively. In contrast, the catalog is declared as a set of book specs. This property does not change over time, at least during the collaboration of selling and buying book between the Bookstore Value Network and the Customer. The Customer seen as whole has a bookshelf and cash. In addition, she keeps in mind the ID number of the book she wants to buy and possibly receives a message describing whether the order she placed was successfully processed or not.

```
lone sig BookCoMarket_C {
        bookstore: one BookstoreValueNetwork_W,
        customer: one Customer_W
}

lone sig BookstoreValueNetwork_W {
        market: one BookCoMarket_C,
        catalog: set BookSpec ,
        inventory: Book set -> Time,
        cash: Int one -> Time
} {
        all t: Time | (int cash.t >= 0)
        market.bookstore = this
        all b: Book, t: Time | b in inventory.t => b.spec in catalog
}

lone sig Customer_W {
        market: one BookCoMarket_C,
        wantedPN: one PartNumber,
        bookshelf: Book set -> Time,
        message: Boolean lone -> Time,
        cash: Int one -> Time
} {
        all t: Time | (int [cash.t] >= 0)
        market.customer = this
}

sig BookSpec { pn: one PartNumber, price: one Int }
```

Attached to the signatures of the Bookstore Value Network and the Customer are unnamed Alloy facts that capture the invariants. For example, the third line of the Alloy

[15] Alloy Analyzer is a tool that can executes code written in the Alloy language http://alloy.mit.edu

fact that is attached to the signature declaring the Bookstore Value Network states that for every book in the inventory, its spec must be in the catalog.

A.1. Declarative Modeling of Local Distributed Action – Net Effect

The local distributed action can be specified as whole in terms of changes made to properties of participating working objects. Figure 52 is the Alloy code that describes the distributed action `sale` by means of an Alloy predicate: `saleAction` with the Alloy keyword `pred`. The two working objects that participate in this action are coded as the first two parameters of the predicate (`aSeller` and `aBuyer`). The last two parameters represent two consecutive moments: before and after the occurrence of the distributed action `sale`.

In the body of predicate `saleAction`, the statements are grouped into invariant, pre-condition and post-condition. The invariant statements describe the logic that is unchanged during the occurrence of the action `sale`: the moment `post` succeeds the moment `pre`, the inventory of the Bookstore Value Network is unchanged before the action `sale` and the book catalog has a book spec that matches the ID number of the book that the Customer wants to buy.

The pre-condition statements say that the inventory contains a book of which spec matches the ID number of the book that the Customer wants to buy before the occurrence of action `sale`. In addition, the Customer must have more cash than the price of the book she wants to buy. Note that the value of the inventory and the cash at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature `Time`).

```
pred saleAction[
   aSeller: one BookstoreValueNetwork_W,      // working object
   aBuyer: one Customer_W,                    // working object
   pre: one Time, post: one Time] {

   // technical invariant
   post = ord/next [pre]
   #aSeller.catalog > 1
   all t: Time, b: Book | pre = ord/next [t] =>
        (b in aSeller.inventory.t <=> b in aSeller.inventory.pre)
   aSeller.segment = aBuyer.segment

   // biz invariant
   one bs: BookSpec | bs.pn = aBuyer.wantedPN and bs in aSeller.catalog

   // pre-condition
   one bk : Book | bk.spec.pn = aBuyer.wantedPN and bk in aSeller.inventory.pre and
        bk not in aBuyer.bookshelf.pre
   one bs: BookSpec | bs.pn = aBuyer.wantedPN and int aBuyer.cash.pre >= int bs.price

   // post-condition
   one bk : Book | bk.spec.pn = aBuyer.wantedPN and
        aSeller.inventory.post = aSeller.inventory.pre - bk and
        aBuyer.bookshelf.post = aBuyer.bookshelf.pre + bk
   one bs: BookSpec | bs.pn = aBuyer.wantedPN and
        int aSeller.cash.post = int aSeller.cash.pre + int bs.price
   one bs: BookSpec | bs.pn = aBuyer.wantedPN and
        int aBuyer.cash.post = int aBuyer.cash.pre - int bs.price
}
```

Figure 52. Alloy code for the `sale` distributed action

The post-condition statements say that a book of which spec matches the ID number of the book that the Customer wants to buy goes from the inventory of the Bookstore Value Network to the bookshelf of the Customer. In addition, the cash of the Bookstore

Value Network is increased by the price of this book while the cash of the Customer is decreased by the same amount. Again, the value of the inventory, the bookshelf and the cash at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature `Time`).

In Figure 53, a) illustrates the state of `Bookstore Value Network` and `Customer` before action `Sale`; b) states after action `Sale`. We can see in this visualization that `Book1` goes from the inventory to the bookshelf and a cash amount of 8 goes from the Customer to the Bookstore Value Network.

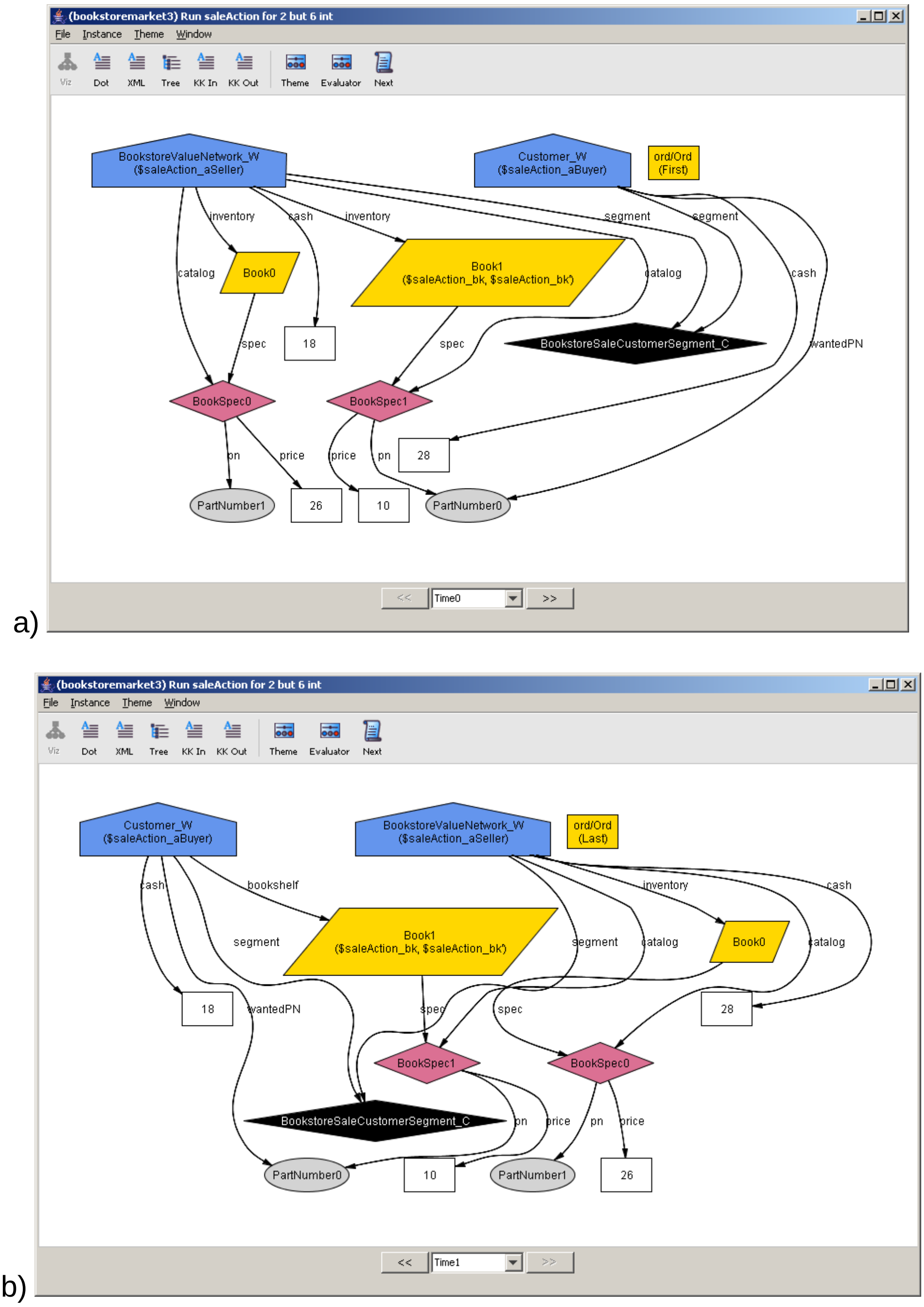

(bookstoremarket3) Run saleAction for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
BookstoreValueNetwork_W ($saleAction_aSeller)
Customer_W ($saleAction_aBuyer)
ord/Ord (First)
inventory
cash
inventory
segment
segment
catalog
Book0
Book1 ($saleAction_bk, $saleAction_bk')
catalog
cash
spec
18
spec
BookstoreSaleCustomerSegment_C
wantedPN
BookSpec0
BookSpec1
pn
price
price
pn
28
PartNumber1
26
10
PartNumber0
<<
Time0
>>
a)
(bookstoremarket3) Run saleAction for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
Customer_W ($saleAction_aBuyer)
BookstoreValueNetwork_W ($saleAction_aSeller)
ord/Ord (Last)
cash
bookshelf
inventory
cash
segment
Book1 ($saleAction_bk, $saleAction_bk')
segment
catalog
Book0
catalog
18
wantedPN
spec
spec
28
BookSpec1
BookSpec0
BookstoreSaleCustomerSegment_C
pn
price
pn
price
PartNumber0
10
PartNumber1
26
<<
Time1
>>
b)

Figure 53. Snapshots showing the two states of `Bookstore Value Network` and `Customer`: before and after the action `Sale`

A.2. Declarative Modeling of Localized Action

The local distributed action can alternatively be distributed into localized actions of participating working objects. Each localized action is specified in terms of change made to the properties of the same working object and some contextual information. The distributed action is then specified as a binding of partial interactions of participating objects. In this binding, localized actions are bound in a declarative way (i.e. the order in which they are bound is not important).

The contextual information needed for the specification of a localized action can be captured in two ways: as an environment of the working object regarding the localized action being specified, or as invariant concepts that should be known by all working objects that participate in the distributed action. These approaches are presented in the following subsections.

A.2.1. One working object and its environment

Figure 54 gives Alloy code for the `Sell` localized action of the Bookstore Value Network. In Alloy, it is encoded as a predicate having the following parameters: `aSeller` (the bookstore), `env` (environment of the bookstore regarding the localized action), `pre` and `post` represent the moments before and after the occurrence of the localized action `Sell`. The environment abstracts away the Customer by representing the spec of the book to order, incoming cash for the payment and the place where the ordered book will leave its inventory.

The statements in the body of the predicate `sellAction` are grouped into invariant, pre-condition and post-condition. The invariant statements describe the logic that is unchanged during the occurrence of the action `Sell`: the moment `post` succeeds the moment `pre`, the inventory of the Bookstore Value Network is unchanged before the action `sale` and the book catalog contains the book spec coming from the environment.

```
pred sellAction[
   aSeller: one BookstoreValueNetwork_W,
   env: one SellEnvironment,
   pre: one Time, post: one Time] {

   // technical invariant
   post = ord/next [pre]
   #aSeller.catalog > 1
   all t: Time, b: Book | pre = ord/next [t] =>
        (b in aSeller.inventory.t <=> b in aSeller.inventory.pre)

   // biz invariant
   env.spec in aSeller.catalog

   // pre-condition
   some bk: aSeller.inventory.pre | bk.spec = env.spec
   one env.in_cash.pre and int env.in_cash.pre = int env.spec.price
   no env.out_book.pre

   // post-condition
   some bk: aSeller.inventory.pre | bk.spec = env.spec and
        aSeller.inventory.post = aSeller.inventory.pre - bk and
               env.out_book.post = bk
   int aSeller.cash.post = int aSeller.cash.pre + int env.spec.price
   no env.in_cash.post
}
```

Figure 54. `Sell` localized action of the `Bookstore Value Network` with environment modeling

The pre-condition statements say that the inventory contains a book of which spec matches the one coming from the environment before the occurrence of action `Sell`. In addition, the cash coming from the environment must be greater or equal the price of the book ordered. Note that the value of the inventory and the cash at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature `Time`).

The post-condition statements say that a book of which spec matches the one from the environment goes from the inventory of the Bookstore Value Network to the environment. In addition, the cash of the Bookstore Value Network is increased by the price of this book while the cash of the environment is decreased by the same amount. Again, the value of the inventory, the book in the environment and the cash at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature `Time`).

Now the two localized actions `Sell` and `Buy` are logically combined to get the semantics of the distributed action `sale` a declarative way as shown in Figure 55. Basically, the three actions are supposed to occur simultaneously (they have the same `pre` and `post` moment). They are combined using the Alloy keyword **and**. In addition, we need to specify that the environment of localized actions `Sell` and that of localized action `Buy` are aligned: they must point to the same book spec.

```
pred saleBinding[store: one BookstoreValueNetwork_W, customer: one Customer_W, pre: one
Time, post: one Time] {
   post = ord/next [pre]
   store.segment = customer.segment

   some s_env: SellEnvironment, b_env: BuyEnvironment |  s_env.spec = b_env.spec and
                     sellAction [store, s_env, pre, post] and
                     buyAction [customer, b_env, pre, post]
}
```

Figure 55. Binding of `Sell` and `Buy` interaction with environment modeling

In Figure 56, a) illustrates the state of `Bookstore Value Network` and `Customer` before the action `Sale` specified as a binding of `Sell` and `Buy` with environment modeling, b) states after action `Sale`. We can see in this visualization that `Book1` goes from the inventory of the Bookstore Value Network to the bookshelf of the Customer via the environment of the localized action `Sell` and a cash amount of 24 goes from the Customer to the Bookstore Value Network.

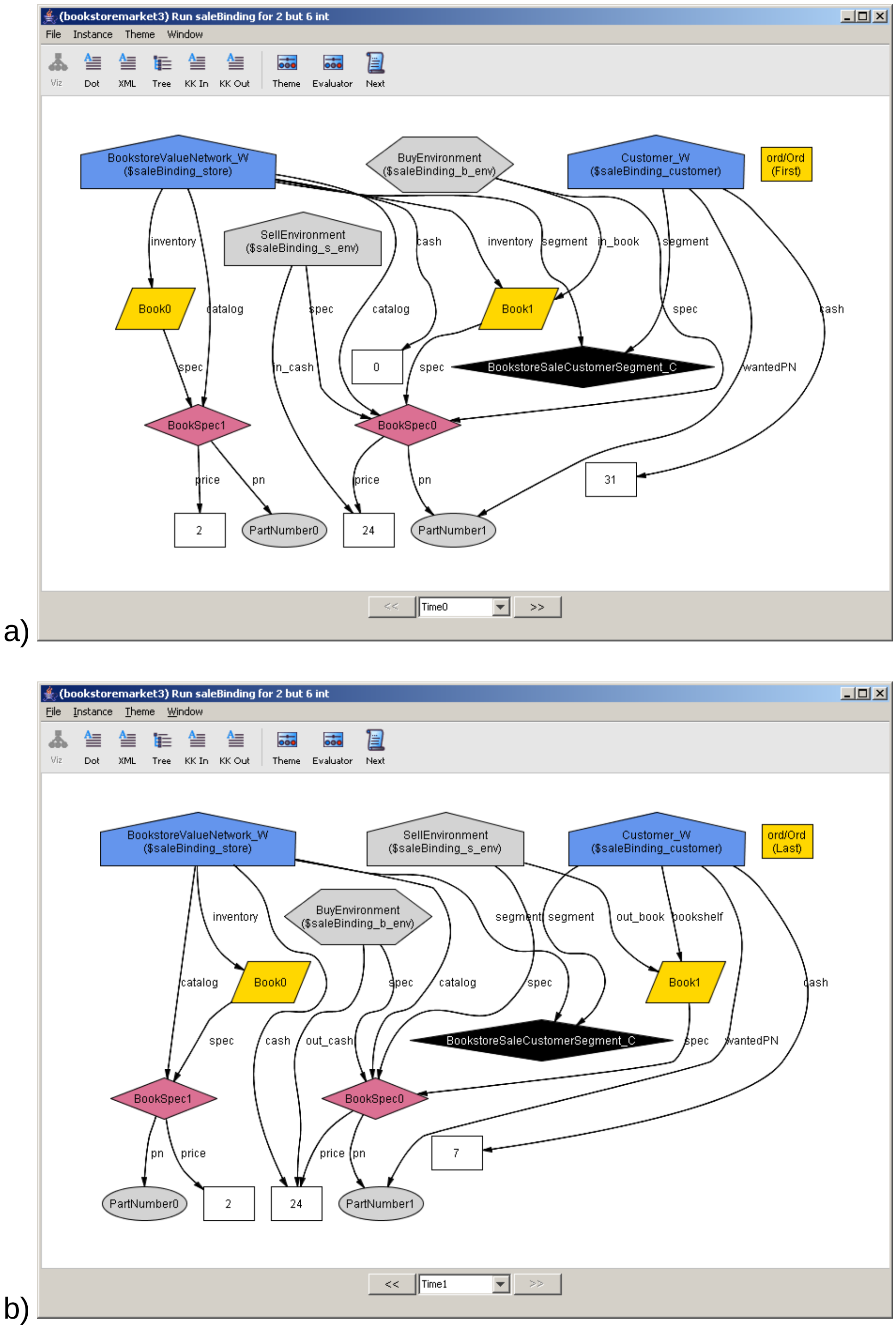

(bookstoremarket3) Run saleBinding for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
BookstoreValueNetwork_W ($saleBinding_store)
BuyEnvironment ($saleBinding_b_env)
Customer_W ($saleBinding_customer)
ord/Ord (First)
SellEnvironment ($saleBinding_s_env)
inventory
cash
inventory
segment
in_book
segment
Book0
catalog
spec
catalog
Book1
spec
cash
spec
in_cash
0
spec
BookstoreSaleCustomerSegment_C
wantedPN
BookSpec1
BookSpec0
price
pn
price
pn
31
2
PartNumber0
24
PartNumber1
<< Time0 >>
a)
(bookstoremarket3) Run saleBinding for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
BookstoreValueNetwork_W ($saleBinding_store)
SellEnvironment ($saleBinding_s_env)
Customer_W ($saleBinding_customer)
ord/Ord (Last)
inventory
BuyEnvironment ($saleBinding_b_env)
segment
segment
out_book
bookshelf
catalog
Book0
spec
catalog
spec
Book1
cash
spec
cash
out_cash
BookstoreSaleCustomerSegment_C
spec
wantedPN
BookSpec1
BookSpec0
pn
price
price
pn
7
PartNumber0
2
24
PartNumber1
<< Time1 >>
b)

Figure 56. Snapshots showing the two states of `BookstoreValueNetwork` and `Customer` with environment modeling: before and after the `Sale` action

A.2.2. Multiple working objects without environment

Figure 57 gives Alloy code for the `Sell` localized action of the Bookstore Value Network without representing the environment. The Alloy predicate of this localized action has the following parameters: `aSeller` (the bookstore), `aBookSpec` / `aBook` (the specification and the book that are transferred between the bookstore and the customer in the `sale` distributed action), `pre` and `post` represent the moments before and after the localized action `Sell`. Note that in the predicate that encodes the `Buy` localized action of the customer, aBookSpec / aBook are also declared as parameters having the same semantics.

The statements in the body of the predicate sellActionWithoutEnv are grouped into invariant, pre-condition and post-condition. The invariant statements describe the logic that is unchanged during the occurrence of the action Sell: the moment post succeeds the moment pre, the inventory of the Bookstore Value Network is unchanged before the action sale and the book catalog contains the book spec that is represented by the parameter aBookSpec.

The pre-condition statements say that the inventory contains a book that is represented by parameter aBook before the occurrence of action Sell. The post-condition statements say that aBook is removed from the inventory of the Bookstore Value Network. In addition, the cash of the Bookstore Value Network is increased by the price of this book. Note that the value of the inventory and the cash at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature Time).

```
pred sellActionWithoutEnv[
   aSeller: one BookstoreValueNetwork_W,
   aBookSpec: one BookSpec,
   aBook: one Book,
   pre: one Time, post: one Time] {

   // invariant
   post = ord/next [pre]
   #aSeller.catalog > 1
   aBook.spec = aBookSpec
   aBookSpec in aSeller.catalog
   all t: Time, b: Book | pre = ord/next [t] =>
        (b in aSeller.inventory.t <=> b in aSeller.inventory.pre)

   // pre-condition
   aBook in aSeller.inventory.pre

   // post-condition
   aSeller.inventory.post = aSeller.inventory.pre - aBook
   int aSeller.cash.post = int aSeller.cash.pre + int aBookSpec.price
}
```

Figure 57. The Sell localized action of the Bookstore Value Network without environment modeling

Now the two localized actions Sell and Buy are combined to specify the distributed action sale in a predicate named saleBindingWithoutEnv as shown in Figure 58. In this predicate, the two variables that represent the book spec and the book are passed as parameters to the two predicates sellActionWithoutEnv and buyActionWithoutEnv together variables that stand for the moments before and after the occurrence of the sale distributed action. In this binding, the two localized actions Sell and Buy are supposed to occur simultaneously.

```
pred saleBindingWithoutEnv[store: one BookstoreValueNetwork_W, customer: one Customer_W,
pre: one Time, post: one Time] {
   post = ord/next [pre]
   store.segment = customer.segment

   some s: BookSpec, book: Book |
            sellActionWithoutEnv [store, s, book, pre, post] and
            buyActionWithoutEnv [customer, s, book, pre, post]
}
```

Figure 58. Binding of Sell and Buy without environment modeling

In Figure 59, a) illustrates the state of `Bookstore Value Network` and `Customer` before the action `Sale` specified as a binding of `Sell` and `Buy` without environment modeling, b) states after action `Sale`. We can see that `Book1` goes from the inventory of the Bookstore Value Network to the bookshelf of the Customer while a cash amount of 1 (the selling price of `Book1`) goes from the Customer to the Bookstore Value Network.

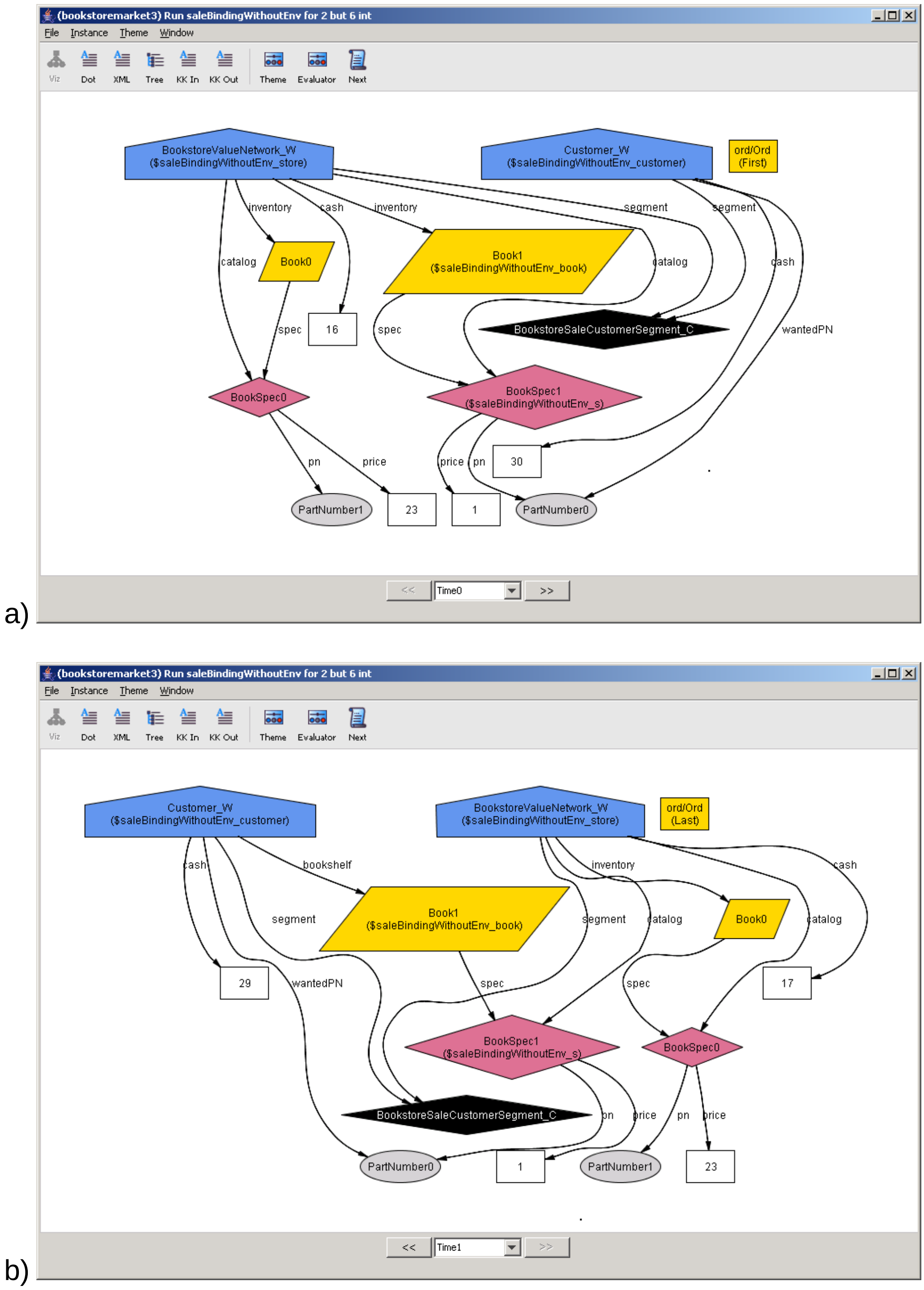

(bookstoremarket3) Run saleBindingWithoutEnv for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
BookstoreValueNetwork_W ($saleBindingWithoutEnv_store)
Customer_W ($saleBindingWithoutEnv_customer)
ord/Ord (First)
inventory
cash
inventory
segment
segment
catalog
Book0
Book1 ($saleBindingWithoutEnv_book)
catalog
cash
spec
16
spec
BookstoreSaleCustomerSegment_C
wantedPN
BookSpec0
BookSpec1 ($saleBindingWithoutEnv_s)
pn
price
price
pn
30
PartNumber1
23
1
PartNumber0
<<
Time0
>>
a)
(bookstoremarket3) Run saleBindingWithoutEnv for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
Customer_W ($saleBindingWithoutEnv_customer)
BookstoreValueNetwork_W ($saleBindingWithoutEnv_store)
ord/Ord (Last)
cash
bookshelf
inventory
cash
segment
Book1 ($saleBindingWithoutEnv_book)
segment
catalog
Book0
catalog
29
wantedPN
spec
spec
17
BookSpec1 ($saleBindingWithoutEnv_s)
BookSpec0
BookstoreSaleCustomerSegment_C
pn
price
pn
price
PartNumber0
1
PartNumber1
23
<<
Time1
>>
b)

Figure 59. Snapshots showing the two states of `BookstoreValueNetwork` and `Customer` with 2 invariant concepts: before and after the `Sale` distributed action

A.3. Declarative Modeling of non-Local Distributed Action – Net Effect

The non-local distributed action can be specified as whole in terms of changes made to properties of participating working objects. Unlike a local distributed action, which does not exchange anything to the environment of the working object in which it is mediated, a non-local distributed action does interact with the environment of the mediating working object. Through this environment, the non-local distributed action takes some input from and/or produces some output to the environment of the mediating working object. These input/output parameters should be the same as those taken or produced by the localize action implemented by the the non-local distributed action being considered. Figure 60 gives Alloy code for the non-local distributed action `marketAndShipAction` between the shipping company and the publishing company that are part of the Bookstore Value Network. Note that the variable `env` represents the environment of the `BookstoreValueNetwork` regarding its localized action `sell` that which is implemented by the distributed action `marketAndShipAction`.

```
pred marketAndShipAction[
   aPublisher: one PublisherCompany_W,
   aShipper: one ShippingCompany_W,
   env: one SellEnvironment,
   pre: one Time, post: one Time] {

   // technical invariant
   post = ord/next [pre]
   #aPublisher.catalog > 1
   all t: Time, b: Book | pre = ord/next [t] =>
        b in aPublisher.inventory.t <=> b in aPublisher.inventory.pre)
   aPublisher.valueNetwork = aShipper.valueNetwork
   all si: ShippingSpec | int env.spec.price > int si.shipping_cost

   // biz invariant
   env.spec in aPublisher.catalog

   // pre-condition
   some bk: aPublisher.inventory.pre | bk.spec = env.spec
        no env.out_book.pre
        one env.in_cash.pre

   // post-condition
   some bk: Book | bk.spec = env.spec and aPublisher.inventory.post =
        aPublisher.inventory.pre - bk and env.out_book.post = bk
   one si: ShippingSpec | int aPublisher.cash.post = int aPublisher.cash.pre +
        int env.spec.price - int si.shipping_cost and
        int aShipper.cash.post = int aShipper.cash.pre + int si.shipping_cost
   no env.in_cash.post
}
```

Figure 60. Alloy code of the non-local distributed action `marketAndShip` between the publishing company and the shipping company

The statements in the body of the predicate `marketAndShipAction` are grouped into invariant, pre-condition and post-condition. The invariant statements describe the logic that is unchanged during the occurrence of the non-local distributed action `marketAndShipAction`: the moment `post` succeeds the moment `pre`, the inventory of the publishing company is unchanged before the action `marketAndShipAction` and the shipping cost must be less than the selling price of the book ordered. Note that for the Customer, the selling price always includes the shipping cost of the book ordered.

The pre-condition statements say that the inventory contains a book that is represented by the book spec referenced by the environment. In addition, the environment must not have any book before the action. The post-condition statements say that a book of which spec matches the book spec referenced by the environment is moved from the inventory of the publishing company to the environment. In addition, the cash of the publishing company is increased by the price of this book minus the shipping cost and the cash of the shipping company is increased by the shipping cost. The cash of the environment disappears. Note that the value of the inventory and the cash at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature `Time`).

Executing predicate `marketAndShipAction` will yield an instance model that is illustrated by Figure 61 a) and Figure 61 b). We can see that `Book1` goes from the inventory of the publishing company to the environment. The selling price of `Book1` is 28. The shipping cost for `Book1` is 4. The cash of the publishing company is increased by an amount of 24 and the cash of the shipping company is increased by an amount of 4. Note that, in Figure 61 a), the cash of the environment is 24 but it disappears in Figure 61 b).

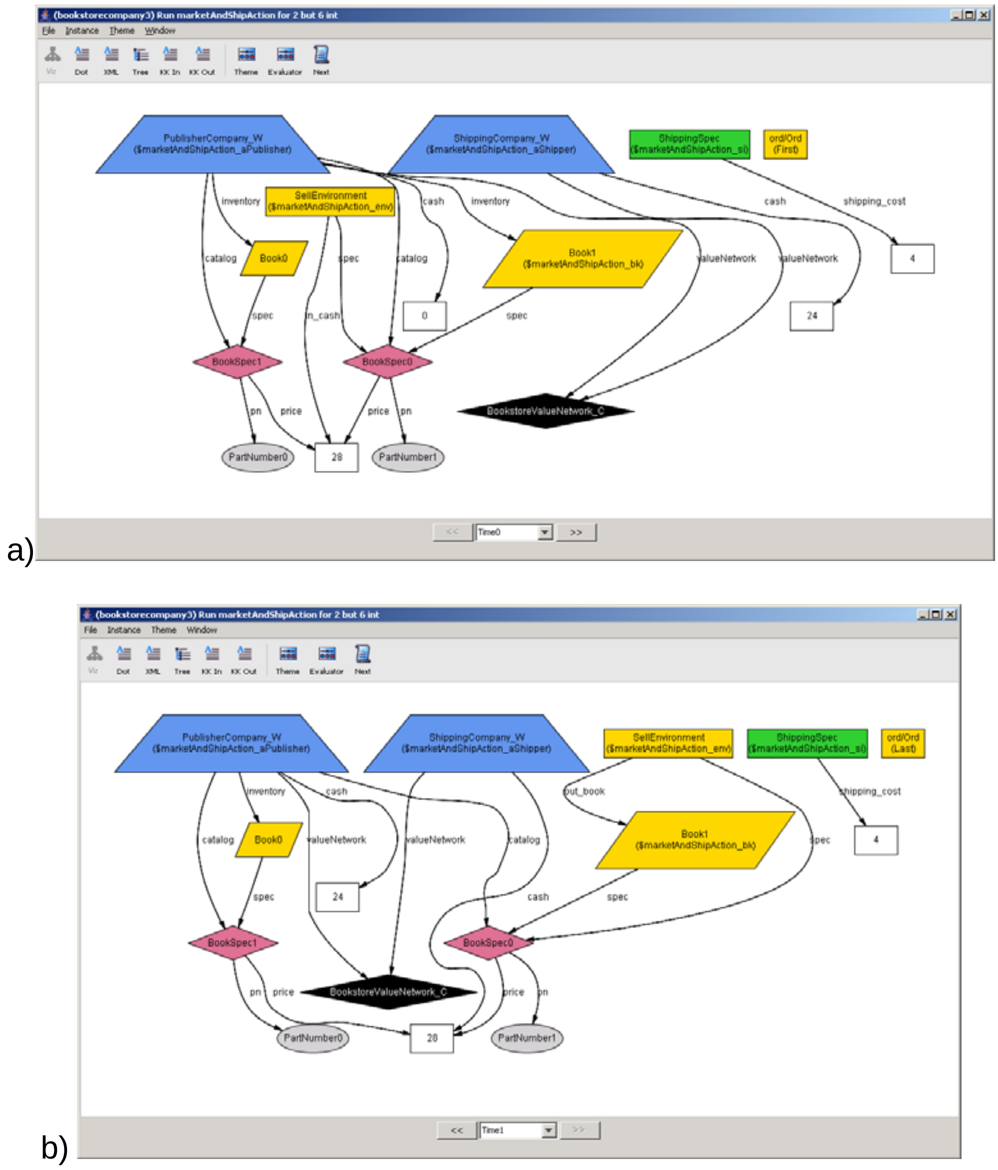

a)
(bookstorecompany3) Run marketAndShipAction for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
PublisherCompany_W ($marketAndShipAction_aPublisher)
ShippingCompany_W ($marketAndShipAction_aShipper)
ShippingSpec ($marketAndShipAction_si)
ord/Ord (First)
inventory
SellEnvironment ($marketAndShipAction_env)
cash
inventory
cash
shipping_cost
catalog
Book0
spec
catalog
Book1 ($marketAndShipAction_bk)
valueNetwork
valueNetwork
4
spec
in_cash
0
spec
24
BookSpec1
BookSpec0
pn
price
price
pn
BookstoreValueNetwork_C
PartNumber0
28
PartNumber1
<<
Time0
>>
b)
(bookstorecompany3) Run marketAndShipAction for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
PublisherCompany_W ($marketAndShipAction_aPublisher)
ShippingCompany_W ($marketAndShipAction_aShipper)
SellEnvironment ($marketAndShipAction_env)
ShippingSpec ($marketAndShipAction_si)
ord/Ord (Last)
inventory
cash
out_book
shipping_cost
catalog
Book0
valueNetwork
valueNetwork
catalog
Book1 ($marketAndShipAction_bk)
spec
4
spec
24
cash
spec
BookSpec1
BookSpec0
pn
price
BookstoreValueNetwork_C
price
pn
PartNumber0
28
PartNumber1
<<
Time1
>>

Figure 61. Snapshots showing the two states of the publishing company and the shipping company: before and after the `marketAndShipAction` specified as net effect

A.4. Declarative Modeling of non-Local Localized Action

The non-local distributed action can alternatively be distributed into localized actions of participating working objects. Each localized action is specified in terms of changes made to the properties of the same working object, the environment of the working object in which it is mediated and some contextual information. The non-local distributed action is then specified as a binding of localized actions of participating objects. Note that localized actions are bound in an unordered way to specify the non-local distributed action.

The contextual information needed for specifying localized actions can be captured in two ways: as environment of the working object regarding the localized action being specified, or as invariant concepts that should be known by all working objects that participate in the distributed action. These approaches are presented in the following subsections.

A.4.1. One working object and its environment

Figure 62 gives Alloy code that encodes the localized action `Market` of the publishing company. Note that there are two variables that represent the environment: `sellEnv` for the environment of the parent working object of the publishing company (the bookstore) regarding the partial interaction `sell`; and `marketEnv` for the environment of the publishing company with respect to the localized action being represented.

```
pred marketAction[
   aPublisher: one PublisherCompany_W,
   sellEnv: one SellEnvironment,
   marketEnv: one MarketEnvironment,
   aShipperInfo: one ShippingSpec,
   pre: one Time, post: one Time] {

   // invariant
   post = ord/next [pre]
   #aPublisher.catalog > 1
   all t: Time, b: Book | pre = ord/next [t] =>
        (b in aPublisher.inventory.t <=> b in aPublisher.inventory.pre)

   // biz invariant
   sellEnv.spec in aPublisher.catalog
   marketEnv.book.spec = sellEnv.spec

   // pre-condition
   one sellEnv.in_cash.pre
   marketEnv.book in aPublisher.inventory.pre

   // post-condition
   aPublisher.inventory.post = aPublisher.inventory.pre - marketEnv.book
   int aPublisher.cash.post = int aPublisher.cash.pre + int sellEnv.spec.price –
        int aShipperInfo.shipping_cost
   no sellEnv.in_cash.post
}
```

Figure 62. Alloy code that specifies the localized action `Market` of the publisher company

A.4.2. Set of working objects with environment

The localized actions of the publishing company and the shipping company can be declaratively combined as shown in Figure 63. Note that the variable `env` that represents the environment of the bookstore is passed to the two predicates that encode the two localized actions `marketAction` and `shipAction`. In addition, there are two environments that are specific to these two ations are also declared: `m_env` and `s_env`. Note that the two predicates take the same parameters that represent the two moments: before and after their occurrence.

```
pred marketAndShipBinding[
   aPublisher: one PublisherCompany_W, aShipper: one ShippingCompany_W,
   env: one SellEnvironment,
   pre: one Time, post: one Time] {

   post = ord/next [pre]
   aPublisher.valueNetwork = aShipper.valueNetwork

   some book: Book, si: ShippingSpec, m_env: MarketEnvironment, s_env: ShipEnvironment |
        marketAction [aPublisher, env, m_env, si, pre, post] and
        shipAction [aShipper, env, s_env, si, pre, post]
}
```

Figure 63. Alloy code of the binding of localized actions performed by the publishing company and the shipping company.

Figure 64 a) and Figure 64 b) are snapshots of the states of the publishing company and the shipping company before and after the action `MarketAndShip`. We can see that `Book1` is moved from the inventory of the publishing company to the field `out_book` of the environment. The cash of the publishing company is increased by an amount of 16, which is equal to subtraction of the selling price of `Book1` by the shipping cost. The cash of the shipping company is increased by amount of 4 - the shipping cost.

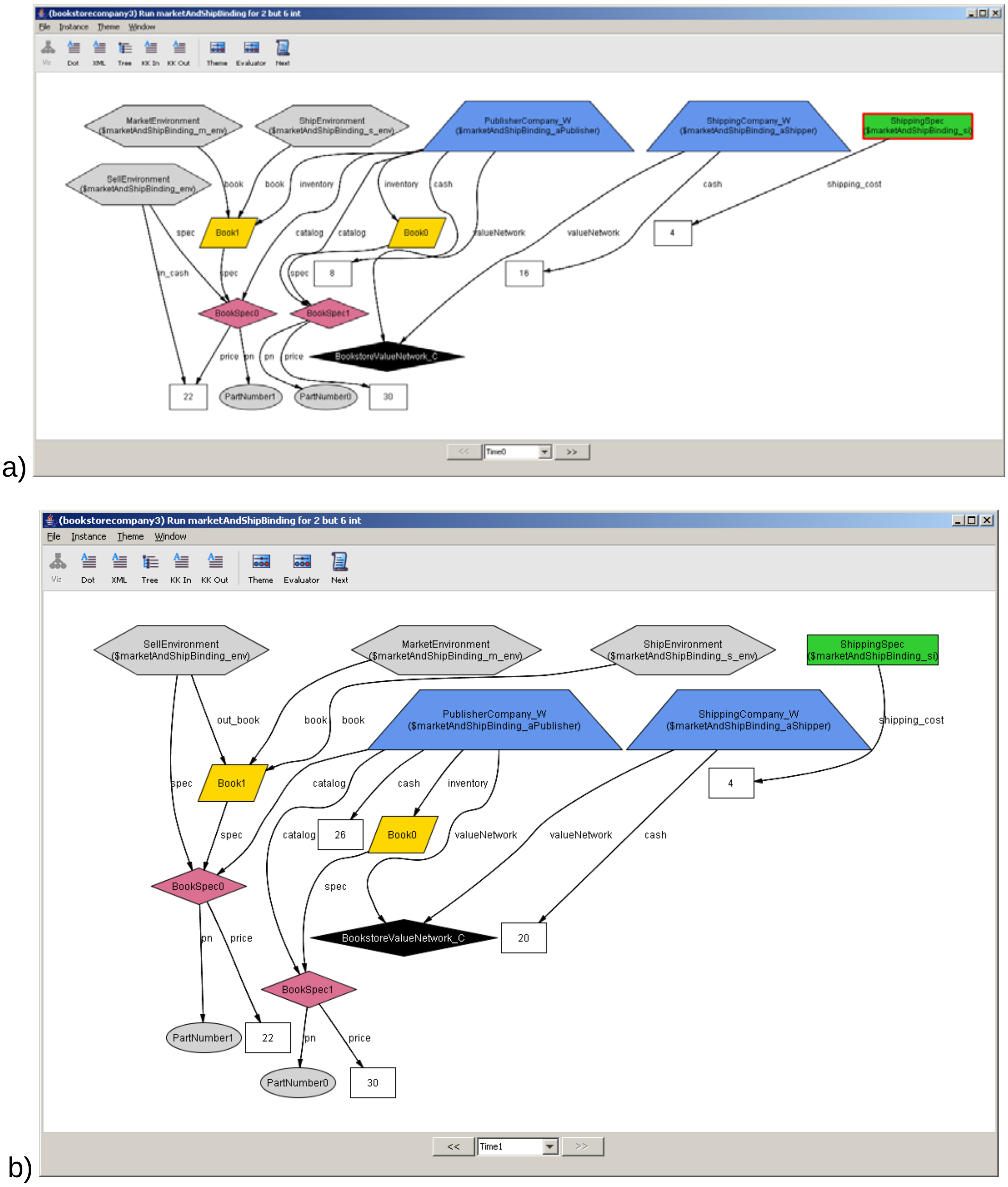

a)
(bookstorecompany3) Run marketAndShipBinding for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
MarketEnvironment ($marketAndShipBinding_m_env)
ShipEnvironment ($marketAndShipBinding_s_env)
PublisherCompany_W ($marketAndShipBinding_aPublisher)
ShippingCompany_W ($marketAndShipBinding_aShipper)
ShippingSpec ($marketAndShipBinding_si)
SellEnvironment ($marketAndShipBinding_env)
book
book
inventory
inventory
cash
cash
shipping_cost
spec
Book1
catalog
catalog
Book0
valueNetwork
valueNetwork
4
in_cash
spec
spec
8
16
BookSpec0
BookSpec1
price
pn
pn
price
BookstoreValueNetwork_C
22
PartNumber1
PartNumber0
30
<<
Time0
>>
b)
(bookstorecompany3) Run marketAndShipBinding for 2 but 6 int
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
SellEnvironment ($marketAndShipBinding_env)
MarketEnvironment ($marketAndShipBinding_m_env)
ShipEnvironment ($marketAndShipBinding_s_env)
ShippingSpec ($marketAndShipBinding_si)
out_book
book
book
PublisherCompany_W ($marketAndShipBinding_aPublisher)
ShippingCompany_W ($marketAndShipBinding_aShipper)
shipping_cost
spec
Book1
catalog
cash
inventory
4
spec
catalog
26
Book0
valueNetwork
valueNetwork
cash
BookSpec0
spec
BookstoreValueNetwork_C
20
pn
price
BookSpec1
PartNumber1
22
pn
price
PartNumber0
30
<<
Time1
>>

Figure 64. Snapshots showing the two states of `PublisherCompany` and `ShippingCompany`: before and after the `MarketAndShip` action specified as a binding

A.5. Imperative Modeling of non-Local Distributed Action

A non-local distributed action can be seen as composite. In this way, it can be considered as an activity combining a set of localized actions performed by participating working objects. The order in this combination is important. In other words, the non-local distributed action is imperatively specified in terms of localized actions performed by participating working objects.

In the bookstore example, the localized action `MarketAndShip` can imperatively be specified by combining the following localized actions

- `prepareBook`: a book that corresponds to the given spec is taken from the inventory of the publisher company
- `deliverBook`: the prepared book is delivered by the shipping company
- `payBook`: the publisher company gets paid
- `payShpping`: the shipping company is paid

A.5.1. One working object with its environment

Figure 65 gives Alloy code of a localized action called `prepareBookAction` performed by the publisher company. The statements in the body of the predicate that declares this action are grouped into invariant, pre-condition and post-condition. The invariant statements describe the logic that is unchanged during the occurrence of the non-local distributed action `prepareBook`: the moment `post` succeeds the moment `pre` and the book catalog contains the book spec that is referenced in the environment.

The pre-condition statements say that the environment does not reference any book and no book is loaded from the inventory yet. The post-condition statements say that a book of which spec matches the book spec referenced by the environment is loaded from the inventory of the publishing company. In addition, the environment still does not reference any book (the book is loaded from the inventory and is ready for further procedures before actually is sent to the environment). Note that the value of the inventory and the reference of the environment at a specific moment can be referenced by a join operation in Alloy (a dot symbol followed by a variable of signature `Time`).

```
pred prepareBookAction[
   aPublisher: one PublisherCompany_W,
   env: one SellEnvironment,
   loadedBook: Book lone -> Time,
   pre: one Time, post: one Time] {

   // invariant
   post = ord/next [pre]
   env.spec in aPublisher.catalog

   // pre-condition
   no env.out_book.pre
   no loadedBook.pre

   // post-condition
   one loadedBook.post
   no env.out_book.post
   loadedBook.post.spec = env.spec
   aPublisher.inventory.post = aPublisher.inventory.pre - loadedBook.post
}
```

Figure 65. Alloy predicate that encodes the action `prepareBook`

A.5.2. Multiple working objects

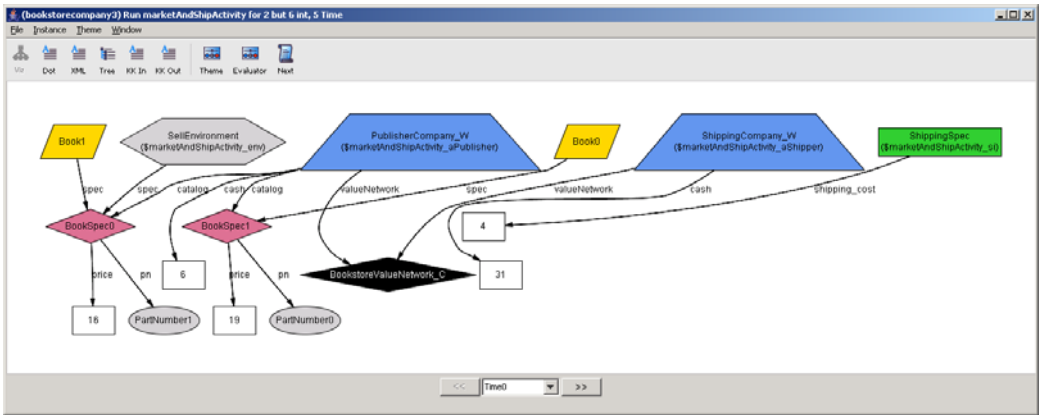
The full distributed action can actually be distributed into localized actions of participating working objects. Each localized action is specified in terms of changes made to the properties of the same working object and changes made to the environment. The full distributed action is then specified as an activity of localize actions performed by participating working objects. In this activity, the order between localized actions is important. In Figure 66, the Alloy predicates that describe how the localized actions `prepareBook`, `deliverBook`, `payBook` and `payShipping` are combined to “implement” the activity of the distributed action `MarketAndShip`. Note that variables `t1`, `t2` and `t3` represent the intermediate moments during the occurrence of the distributed action `MarketAndShip`. They define the moments that one of these localized actions finishes and another localized action is about to occur.

The activity of the distributed action `MarketAndShip` can be interpreted as follows. The `prepareBookAction` localized action loads a book (that corresponds to the book spec given by the environment) from the inventory of the publishing company. The `deliverBookAction` localized action puts the loaded book to the environment. The `payBookAction` localized action pays the publishing company and the `payShippingAction` localized action pays the shipping company.

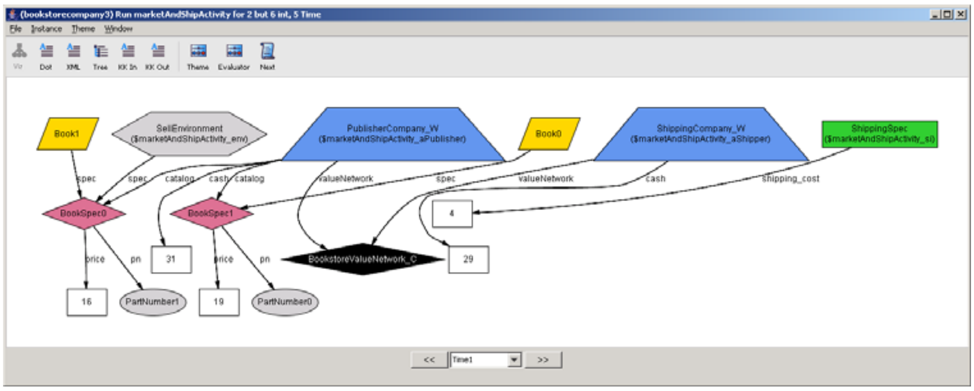
```
pred marketAndShipActivity[
   aPublisher: one PublisherCompany_W,
   aShipper: one ShippingCompany_W,
   env: one Environment,
   pre: one Time, post: one Time] {

   some t1, t2, t3: Time | some loadedBook: Book lone -> Time | some si: ShippingSpec |
        t1 = ord/next [pre] and t2 = ord/next [t1] and t3 = ord/next [t2] and post =
ord/next [t3] and
        int env.spec.price > int si.shipping_cost and
        prepareBookAction[aPublisher, env, loadedBook, pre, t1] and
        deliverBookAction[aShipper, env, loadedBook, t1, t2] and
        payBookAction[aPublisher, env, si, t2, t3] and
        payShippingAction[aShipper, env, si, t3, post] and
        no env.in_cash.post
}
```

Figure 66. Ordered combination of localized actions performed by `PublisherCompany` and `ShippingCompany` that makes up `MarketAndShip` up.

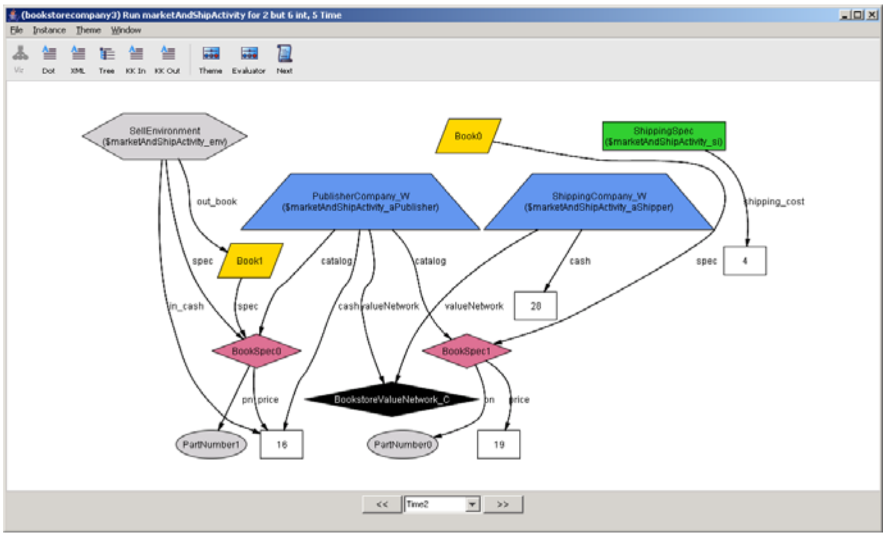
Figure 67 a), Figure 64 b), Figure 64 c), Figure 64 d) and Figure 64 e) are snapshots of the states of the publishing company and the shipping company before action `MarketAndShip`, after action `prepareBook`, after action `deliverBook`, after action `payBook` and after action `MarketAndShip`.


(bookstorecompany3) Run marketAndShipActivity for 2 but 6 int, 5 Time

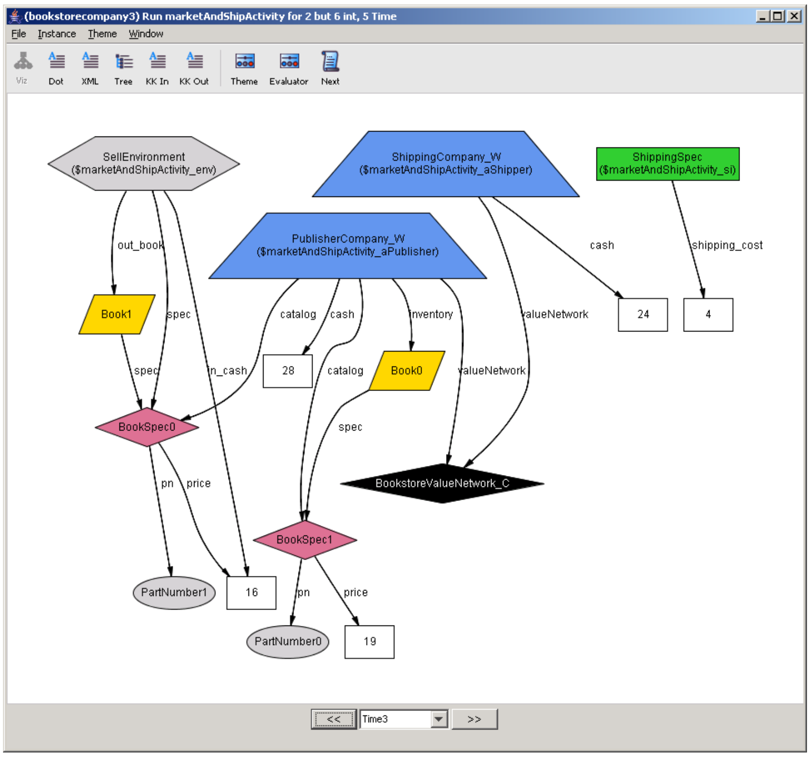
a)


(bookstorecompany3) Run marketAndShipActivity for 2 but 6 int, 5 Time
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
Book1
SellEnvironment ($marketAndShipActivity_env)
PublisherCompany_W ($marketAndShipActivity_aPublisher)
Book0
ShippingCompany_W ($marketAndShipActivity_aShipper)
ShippingSpec ($marketAndShipActivity_si)
spec
spec
catalog
cash
catalog
valueNetwork
spec
valueNetwork
cash
shipping_cost
BookSpec0
BookSpec1
4
price
pn
31
price
pn
BookstoreValueNetwork_C
29
16
PartNumber1
19
PartNumber0
<<
Time1
>>

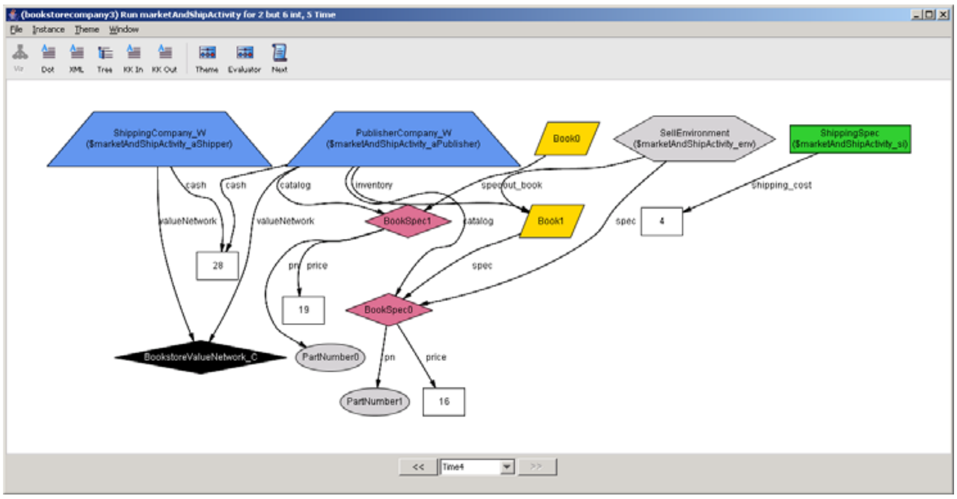
b)


(bookstorecompany3) Run marketAndShipActivity for 2 but 6 int, 5 Time
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
SellEnvironment ($marketAndShipActivity_env)
Book0
ShippingSpec ($marketAndShipActivity_si)
out_book
PublisherCompany_W ($marketAndShipActivity_aPublisher)
ShippingCompany_W ($marketAndShipActivity_aShipper)
shipping_cost
spec
Book1
catalog
catalog
cash
spec
4
in_cash
spec
cash
valueNetwork
valueNetwork
28
BookSpec0
BookSpec1
pn
price
BookstoreValueNetwork_C
pn
price
PartNumber1
16
PartNumber0
19
<<
Time2
>>

c)


(bookstorecompany3) Run marketAndShipActivity for 2 but 6 int, 5 Time
File Instance Theme Window
Viz Dot XML Tree KK In KK Out Theme Evaluator Next
SellEnvironment ($marketAndShipActivity_env)
ShippingCompany_W ($marketAndShipActivity_aShipper)
ShippingSpec ($marketAndShipActivity_si)
out_book
PublisherCompany_W ($marketAndShipActivity_aPublisher)
cash
shipping_cost
Book1
spec
catalog
cash
inventory
valueNetwork
24
4
spec
in_cash
28
catalog
Book0
valueNetwork
BookSpec0
spec
pn
price
BookstoreValueNetwork_C
BookSpec1
PartNumber1
16
pn
price
PartNumber0
19
<<
Time3
>>

d)


(bookstorecompany3) Run marketAndShipActivity for 2 but 6 int, 5 Time
File
Instance
Theme
Window
Viz
Dot
XML
Tree
KK In
KK Out
Theme
Evaluator
Next
ShippingCompany_W ($marketAndShipActivity_aShipper)
PublisherCompany_W ($marketAndShipActivity_aPublisher)
Book0
SellEnvironment ($marketAndShipActivity_env)
ShippingSpec ($marketAndShipActivity_si)
cash
cash
catalog
inventory
spec
out_book
shipping_cost
valueNetwork
valueNetwork
BookSpec1
catalog
Book1
spec
4
28
pn
price
spec
19
BookSpec0
BookstoreValueNetwork_C
PartNumber0
pn
price
PartNumber1
16
<<
Time4
>>

e)

Figure 67. Snapshots showing the 5 states of `BookstoreValueNetwork` and `Customer` with environment concepts: initial, book prepared, book delivered, book payment and shipment payment.

It is possible to map the way that a full distributed action is distributed into localized actions of participating working objects to BPEL[16]. Each localized action can be mapped to a service provided by the working object in which it is defined. This mapping is feasible if BPEL is extended to include human activities and services of non-IT entity [43]. If the working object is an IT system, its localized actions should be regarded as web services. The full distributed action is then coded in BPEL using BPEL constructs. For example, localized actions are called using BPEL invoke; the order between these localized actions can be captured using the sequence or the if-then-else construct of BPEL.

[16] Business Process Execution Language http://www.bpelsource.com

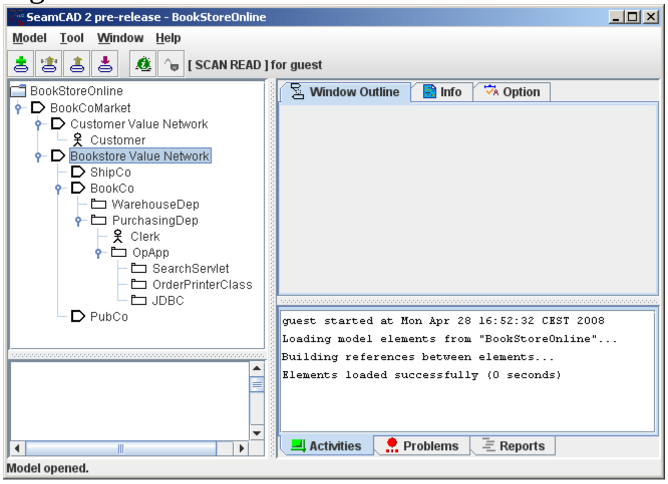
Appendix B: Tutorial and Questionnaire Used for Obtaining Feedback on SeamCAD

This appendix includes material used for working with practitioners and students who validated SeamCAD. Section B.1 and B.2 present the tutorial that provided the practitioners with step-by-step scenarios for getting familiar with the SeamCAD modeling language and the computer-aided tool. Section B.3 gives the slides that provided students with instructions of how to validate SeamCAD. Secion B.4 and B.5 are the questionnaire used for getting feedback from practitioners and students, respectively.

B.1. Tutorial: Viewing a pre-Built Model in SeamCAD

Navigate in an existing model of a bookstore that goes online. Diagrams are opened to shows different organizational levels and functional levels.

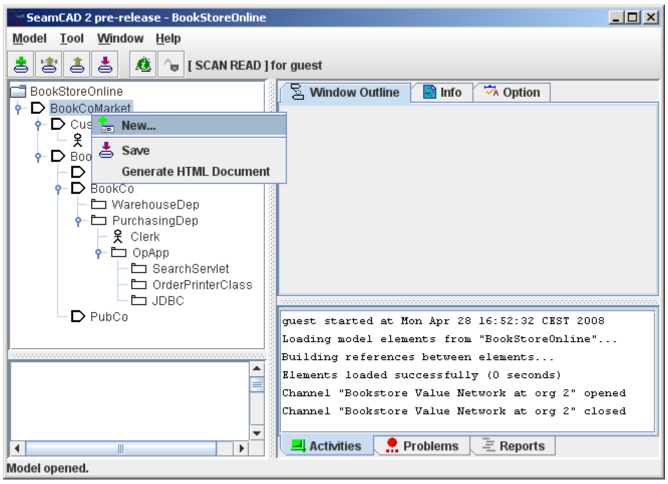
1. Download the jar file `SeamCAD.jar` from seamcad.epfl.ch to your working directory.
2. Type `java -jar SeamCAD.jar` to launch SeamCAD. You will get the login window of the tool.
3. In the login window, make sure that the checkbox “As Local Guest” is checked, then hit the button “Login”.
4. In the main window, select menu “Model” > “Open” or an equivalent button of the toolbar.
5. Open the existing model `BookstoreOnline`. Looking at the overview of the model in the tree-view at the top-left corner of the main window to count the number of the organizational levels.



```
guest started at Mon Apr 28 16:52:32 CEST 2008
Loading model elements from "BookStoreOnline"...
Building references between elements...
Elements loaded successfully (0 seconds)
```

Model opened.

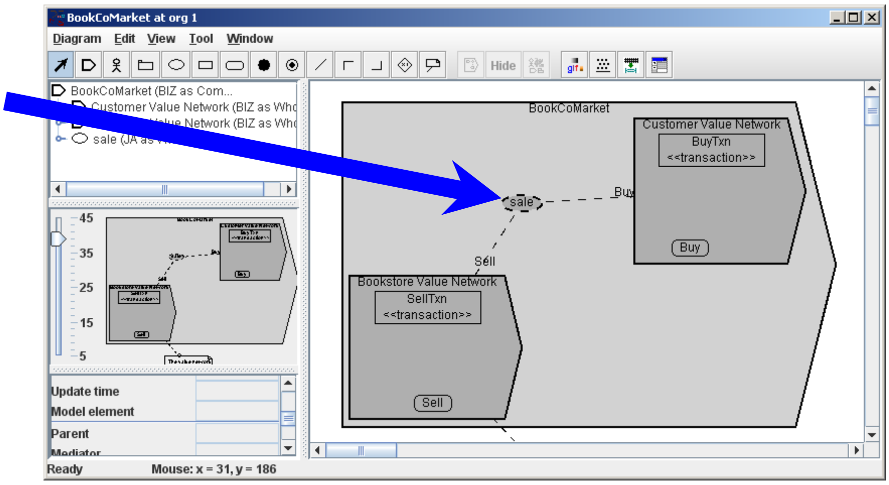
6. Open an editing window to show a certain organizational level (right click on the top object `BookCoMarket` in the tree-view and select menu “New…”)



```
guest started at Mon Apr 28 16:52:32 CEST 2008
Loading model elements from "BookStoreOnline"...
Building references between elements...
Elements loaded successfully (0 seconds)
Channel "Bookstore Value Network at org 2" opened
Channel "Bookstore Value Network at org 2" closed
```

7. Explore two different functional levels in the first organizational level
   - Double-click on the action `sale` to get to the second functional level. Component actions of `sale` are now visible. Accordingly, properties and localized actions of the two value networks are shown in details.

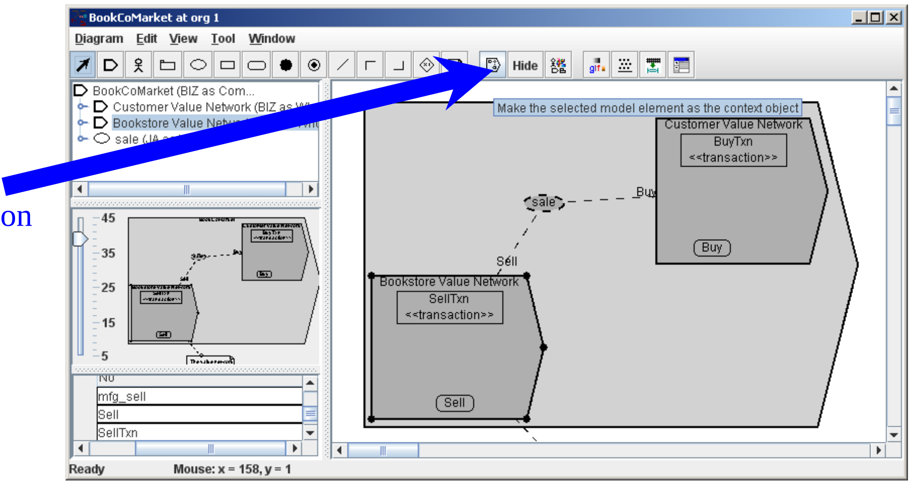
Double-click this pictogram



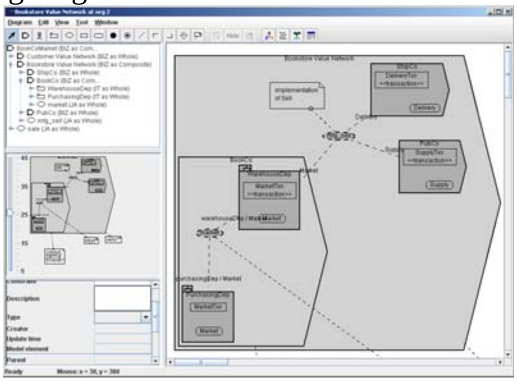
   - Double-click on the action `sale` again to get back to the first functional level.

8. Make sure that `Bookstore Value Network` is selected in the diagram and click a button in the toolbar to jump to the second organizational level. In this organizational level, three companies `BookCo`, `ShipCo` and `PubCo` should be visible.

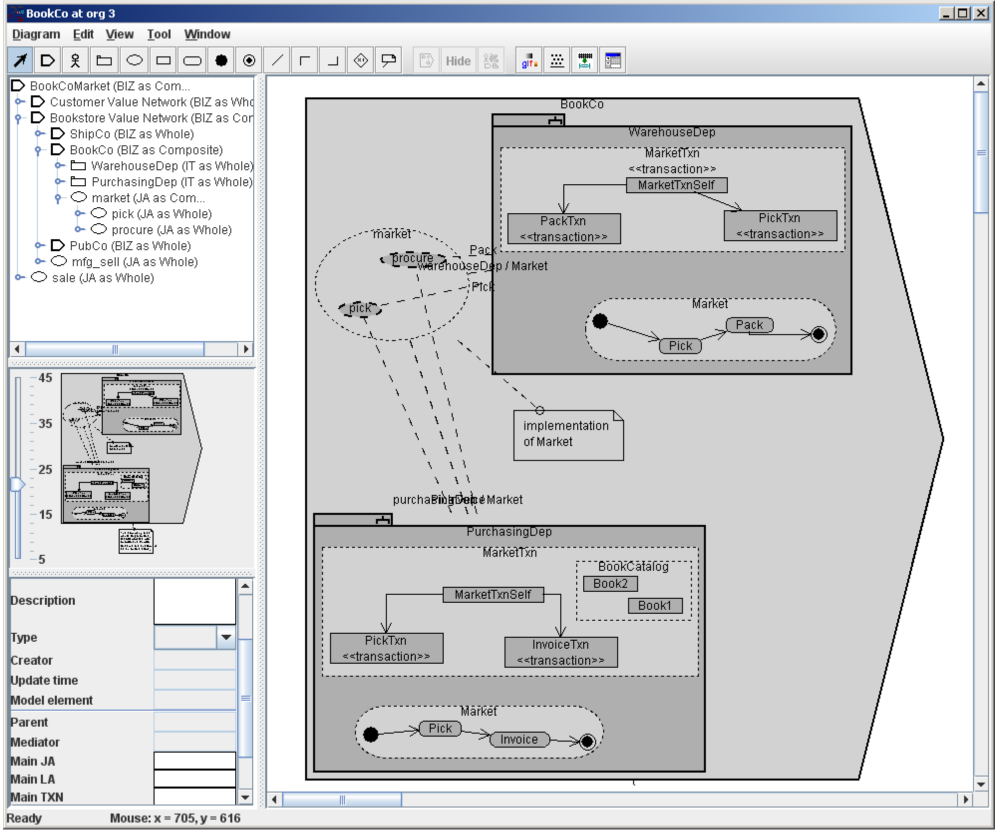
Click this button



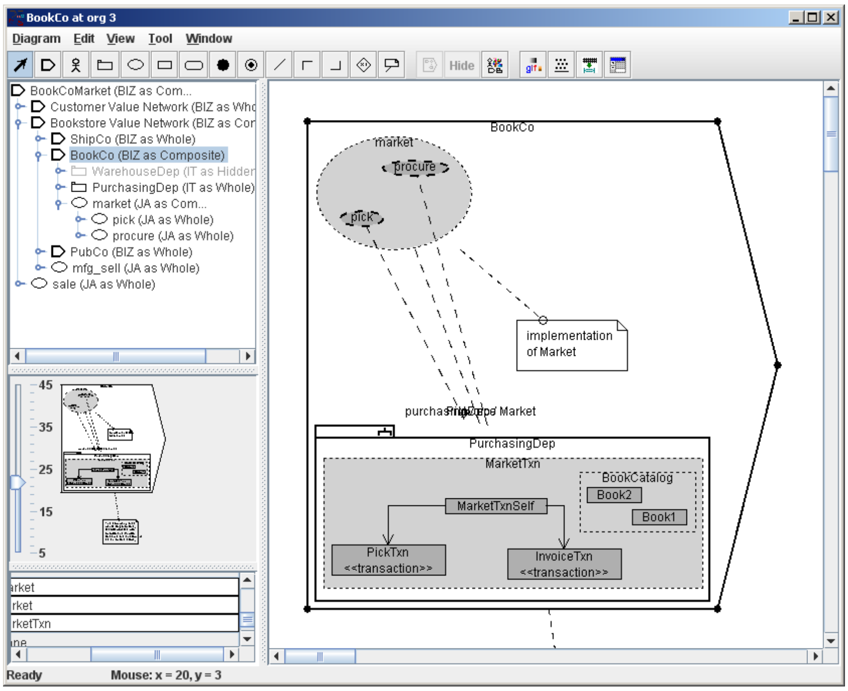
9. Double-click on the pictogram of `BookCo` to go deeper to the third organizational level. The following diagram should be obtained



10. Right-click on the pictogram of `BookCo` and select “as Context” get focused in `BookCo` and its internal structure. Explore two different functional levels in this organizational level and experience the show/hide functionality
    - Double-click on the action `market` to get to the second functional level. Double-click on `BookCatalog` to reveal its component properties.



    - Select the `WarehouseDep` and click on the button “Hide” of the toolbar to make this object invisible in the diagram. Right-click on the pictogram `PurchasingDep` and select menu “Hide behavior”, you will get the following diagram



B.2. Tutorial: Adding more Model Elements…

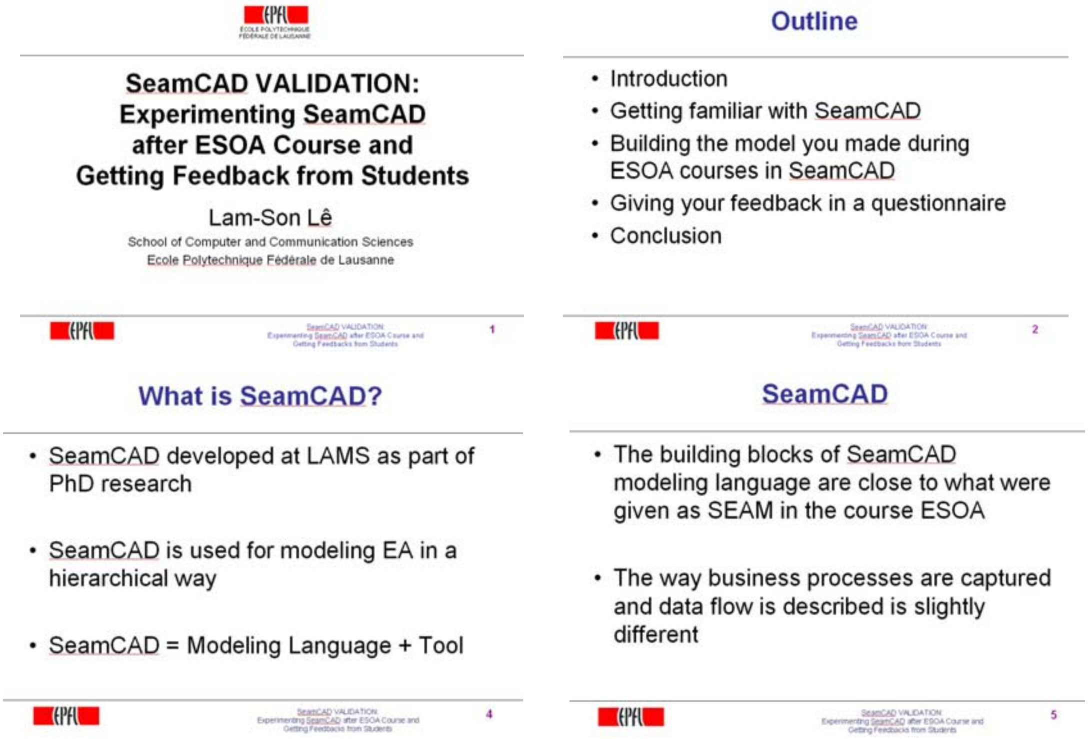
You have learnt how to navigate in an existing model. The internal structure of `BookCo` has been defined in the model, but `ShipCo` and `PubCo` are still empty. Now, additional model elements can be created to define the internal structure of a company, say `ShipCo`.

11. Switch to the main window, right-click on the tree node `ShipCo` and select menu "as Context…", you get a new editing window where only `ShipCo` is visible in its diagram.
12. Insert departments (using block arrow or subsystem pictogram) or people (using the stickman pictogram) that constitute the `ShipCo` in the diagram.
13. Insert an action using ellipse pictogram inside the pictogram of `ShipCo` and name it `ship`. Double-click on this action to view it as composite.
14. Insert two more actions within action `ship`, namely `handle` and `deliver`.
15. Create participation links by connecting departments and people to newly-created actions
16. Move around the automatically-created transactions and localized actions to your preference
17. Insert start symbol and stop symbol in localized actions that are already viewed as composite
18. Make transition sequences between localized actions

B.3. Slides: Building Enterprise Model in SeamCAD

A total of 9 master's students in our university participated in a one-day validation session of SeamCAD. They all took a master's course titled "Enterprise and Service Oriented Architecture" given by our group in the university. In this course, the whole class was divided into small groups each of whom built an imaginary company to manufacture and sell diesel-powered aircraft engines. Each group developed an enterprise model for their company on pieces if paper following the SEAM method. Their enterprise models cover several organizational levels: value network, company and IT application.

In SeamCAD validation, participating students first got familiar to the SeamCAD modeling language and tool by creating a model that represent the organizational structure of our university. Then they were asked to rebuild their paper-based enterprise model in the SeamCAD tool. Next, they reviewed their model by browsing it opening different diagrams that show different organizational levels and functional levels of the model. Finally, they fill in a questionnaire to give their feedback. On the average, it took 7 hours for each student to complete the validation. Figure 68 lists the main slides that were used for working with these students: Figure 68 a) – slides that gave some opening remarks, Figure 68 b) – slides that introduced SeamCAD, Figure 68 c) – slides that guided the students in building and reviewing their enterprise model in SeamCAD.



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedback from Students

Lam-Son Lê

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne

SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 1

Outline

- Introduction
- Getting familiar with SeamCAD
- Building the model you made during ESOA courses in SeamCAD
- Giving your feedback in a questionnaire
- Conclusion

SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 2

What is SeamCAD?

- SeamCAD developed at LAMS as part of PhD research
- SeamCAD is used for modeling EA in a hierarchical way
- SeamCAD = Modeling Language + Tool

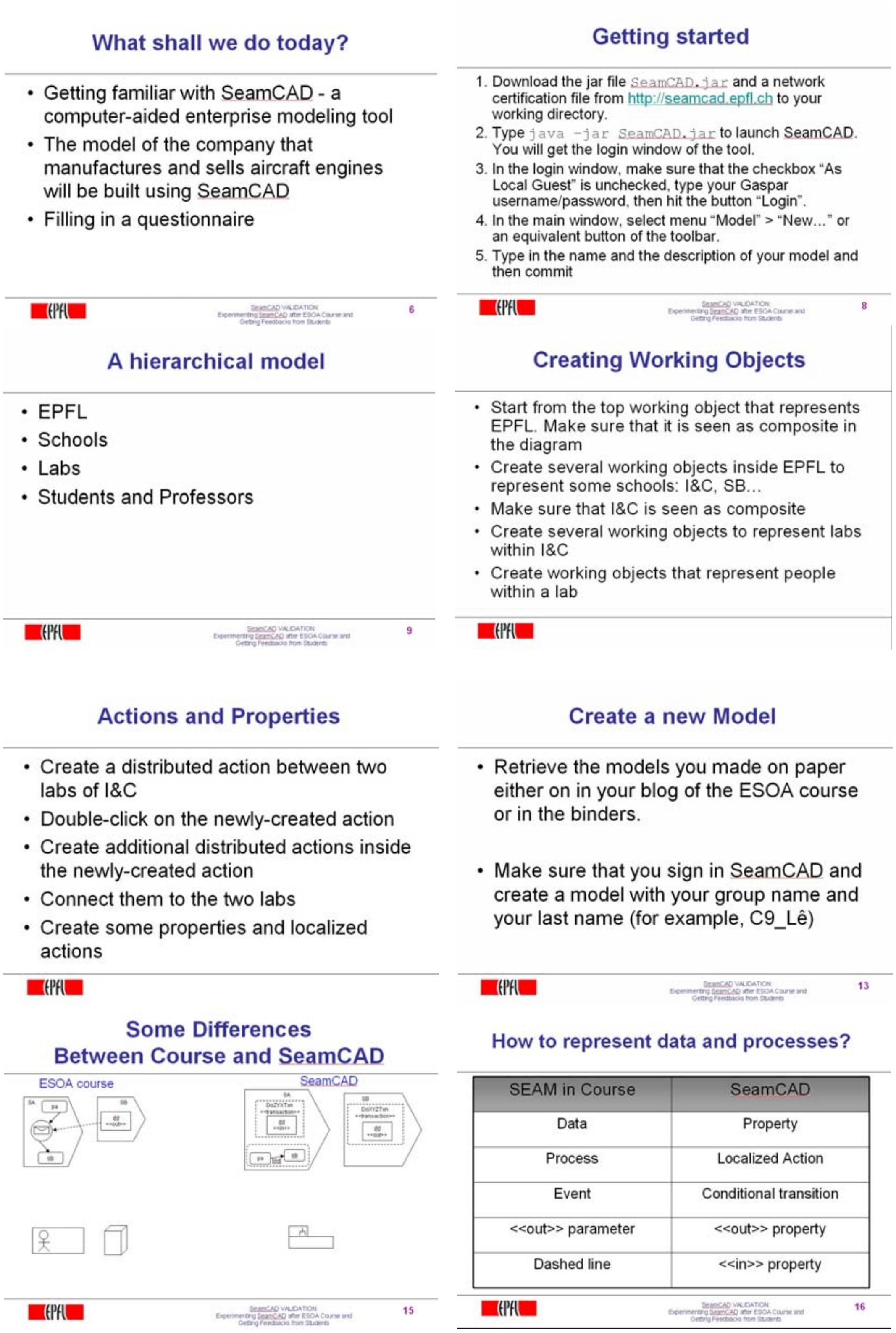
SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 4

SeamCAD

- The building blocks of SeamCAD modeling language are close to what were given as SEAM in the course ESOA
- The way business processes are captured and data flow is described is slightly different

SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 5

Figure 68 a). Opening remarks for working with master’s students to validate SeamCAD



What shall we do today?

- Getting familiar with SeamCAD - a computer-aided enterprise modeling tool
- The model of the company that manufactures and sells aircraft engines will be built using SeamCAD
- Filling in a questionnaire

SeamCAD VALIDATION
Experimenting SeamCAD after ESOA Course and
Getting Feedbacks from Students
6

Getting started

1. Download the jar file `SeamCAD.jar` and a network certification file from http://seamcad.epfl.ch to your working directory.
2. Type `java -jar SeamCAD.jar` to launch SeamCAD. You will get the login window of the tool.
3. In the login window, make sure that the checkbox "As Local Guest" is unchecked, type your Gaspar username/password, then hit the button "Login".
4. In the main window, select menu "Model" > "New…" or an equivalent button of the toolbar.
5. Type in the name and the description of your model and then commit

SeamCAD VALIDATION
Experimenting SeamCAD after ESOA Course and
Getting Feedbacks from Students
8

A hierarchical model

- EPFL
- Schools
- Labs
- Students and Professors

SeamCAD VALIDATION
Experimenting SeamCAD after ESOA Course and
Getting Feedbacks from Students
9

Creating Working Objects

- Start from the top working object that represents EPFL. Make sure that it is seen as composite in the diagram
- Create several working objects inside EPFL to represent some schools: I&C, SB…
- Make sure that I&C is seen as composite
- Create several working objects to represent labs within I&C
- Create working objects that represent people within a lab

Actions and Properties

- Create a distributed action between two labs of I&C
- Double-click on the newly-created action
- Create additional distributed actions inside the newly-created action
- Connect them to the two labs
- Create some properties and localized actions

Create a new Model

- Retrieve the models you made on paper either on in your blog of the ESOA course or in the binders.
- Make sure that you sign in SeamCAD and create a model with your group name and your last name (for example, C9_Lê)

SeamCAD VALIDATION
Experimenting SeamCAD after ESOA Course and
Getting Feedbacks from Students
13

Some Differences Between Course and SeamCAD

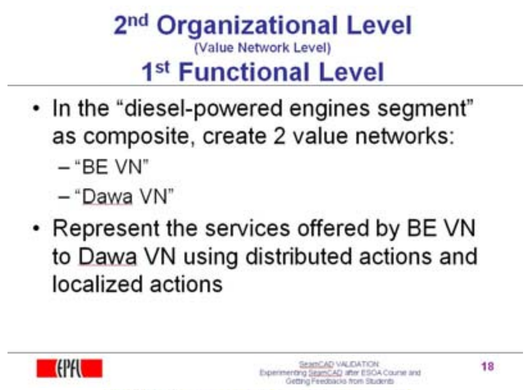
SeamCAD VALIDATION
Experimenting SeamCAD after ESOA Course and
Getting Feedbacks from Students
15

How to represent data and processes?

SEAM in Course	SeamCAD
Data	Property
Process	Localized Action
Event	Conditional transition
<<out>> parameter	<<out>> property
Dashed line	<<in>> property

SeamCAD VALIDATION
Experimenting SeamCAD after ESOA Course and
Getting Feedbacks from Students
16

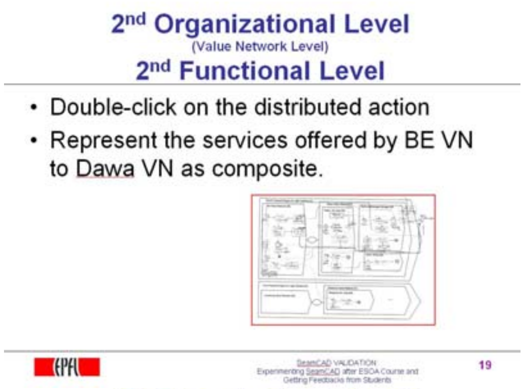
Figure 68 b). Introduction of SeamCAD to students



2nd Organizational Level
(Value Network Level)
1st Functional Level

- In the “diesel-powered engines segment” as composite, create 2 value networks:
  - “BE VN”
  - “Dawa VN”
- Represent the services offered by BE VN to Dawa VN using distributed actions and localized actions

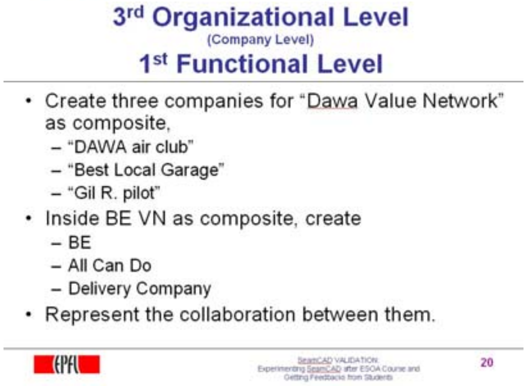
SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 18



2nd Organizational Level
(Value Network Level)
2nd Functional Level

- Double-click on the distributed action
- Represent the services offered by BE VN to Dawa VN as composite.

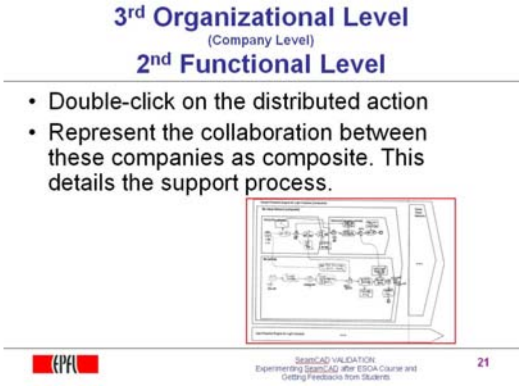
SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 19



3rd Organizational Level
(Company Level)
1st Functional Level

- Create three companies for “Dawa Value Network” as composite,
  - “DAWA air club”
  - “Best Local Garage”
  - “Gil R. pilot”
- Inside BE VN as composite, create
  - BE
  - All Can Do
  - Delivery Company
- Represent the collaboration between them.

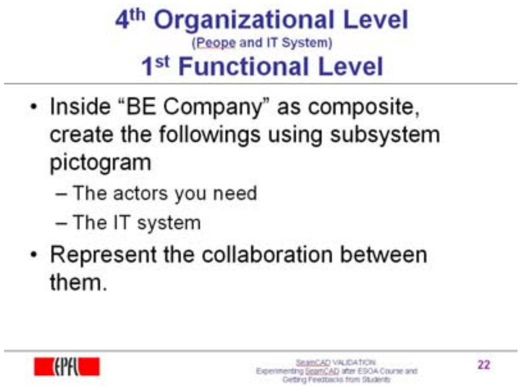
SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 20



3rd Organizational Level
(Company Level)
2nd Functional Level

- Double-click on the distributed action
- Represent the collaboration between these companies as composite. This details the support process.

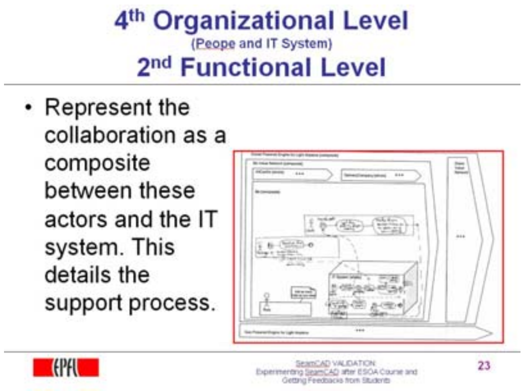
SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 21



4th Organizational Level
(Peope and IT System)
1st Functional Level

- Inside “BE Company” as composite, create the followings using subsystem pictogram
  - The actors you need
  - The IT system
- Represent the collaboration between them.

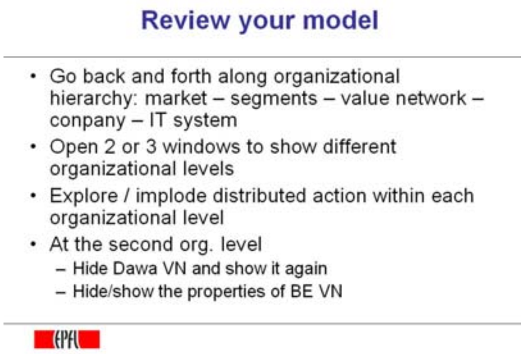
SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 22



4th Organizational Level
(Peope and IT System)
2nd Functional Level

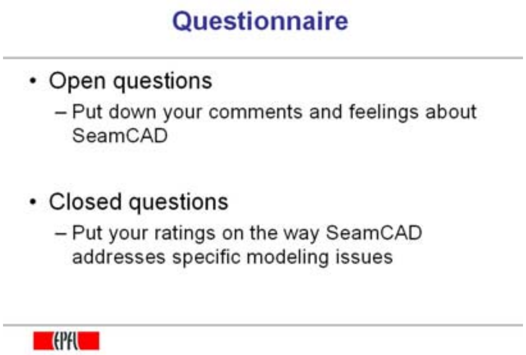
- Represent the collaboration as a composite between these actors and the IT system. This details the support process.

SeamCAD VALIDATION: Experimenting SeamCAD after ESOA Course and Getting Feedbacks from Students 23



Review your model

- Go back and forth along organizational hierarchy: market – segments – value network – conpany – IT system
- Open 2 or 3 windows to show different organizational levels
- Explore / implode distributed action within each organizational level
- At the second org. level
  - Hide Dawa VN and show it again
  - Hide/show the properties of BE VN



Questionnaire

- Open questions
  - Put down your comments and feelings about SeamCAD
- Closed questions
  - Put your ratings on the way SeamCAD addresses specific modeling issues

Figure 68 c). Instructions on how to build and review
an enterprise model that was previously made on paper in SeamCAD

B.4. Questionnaire – Obtaining Feedback from Practitioners and Researchers

Name:

Company: Date:

SeamCAD Validation QUESTIONNAIRE

Introduction

1. Do you use personally or do you have people in your organization developing hierarchical or enterprise model? Please detail (who is using modeling, to do what, which tool is used, etc…).

2. In your organization, do you model enterprise systems across organizational level?

   organizational levels describe the structure of business entities, for example value network composed of companies, companies composed of people and IT system, IT system composed of applications.

   a. If yes, what organizational levels do you consider?

   b. If no, why? Would you believe that organizational levels bring added value?

3. In your organization, is functional level used?

   functional levels describe the behavior of business entities at different level of granularity, for example a “sale” action can be broken down into an activity composed of “get order”, “pay” and “deliver”; “get order” can itself be broken down into “show catalog” and “get book id”.

   a. If yes, what functional levels do you have?

   b. If no, why? Would you believe that functional levels bring added value?

4. In your organization, do you analyze and design the structure of multiple “organizations” at the same time?

   for example, analyzing how two companies are organized internally to implement a business process between them.

   a. If yes, please detail (in which context do you do it and how?)

   b. If no, would it be useful to have such feature?

5. Which approach do you follow in building your enterprise models in your practice?

    a. Top-down b. Bottom-up c. None d. Both

6. What notation do you use in your practice?
    could be your own notation or some well-known notation such as BPMN, UML, informal business notation…

Organizational level

7. How do you rate the way SeamCAD manages organizational levels?
    how the model and diagrams can be browsed by going back and forth between, for example, value network, company level and IT level

    a. excellent
    b. good
    c. bad
    d. very bad

    Any suggestion?

8. Do you think that the top-down approach of SeamCAD has some value in your practice? Please explain in details

    a. Yes b. No

    Explanation:

Functional level

9. How do you rate the way SeamCAD manages functional levels?
    the level of granularity of behavior in a diagram can be changed easily, for example, from “sale” action to three smaller actions “get order”, “pay” and “deliver”

    a. excellent
    b. good
    c. bad
    d. very bad

    Any suggestion?

Multi-system

10. How do you rate the way SeamCAD handles multiple system representation?

    a. excellent
    b. good
    c. bad
    d. very bad

    Any suggestion?

Model coherence and Diagrams

11. How do you rate the feature of SeamCAD through which diagrams can be opened as partial views of a common model?

    A diagram in SeamCAD can be opened by choosing a specific object as the context object.
    The context object has the outermost pictogram in the diagram.

    a. excellent
    b. good
    c. bad
    d. very bad

    Any suggestion?

12. How do you rate the way diagrams can be customized by hiding / showing specific systems and actions in SeamCAD?

    a. excellent
    b. good
    c. bad
    d. very bad

    Any suggestion?

Notation

13. How do you rate the intuitiveness of the notation scheme used in SeamCAD?

    a. excellent
    b. good
    c. bad
    d. very bad

    Any suggestion?

General issues

14. What features of SeamCAD do you like most?

15. What features of SeamCAD do you consider as unnecessary?

16. How do you rate the user-friendliness of SeamCAD

    a. excellent
    b. good
    c. bad
    d. very bad

Any suggestion?

17. How would you classify SeamCAD?

    a. Computer-Aided Enterprise Modeling tool
    b. Computer-Aided Requirement Engineering tool
    c. Computer-Aided Design tool
    d. I propose the term ……………………………………………..………..…….

B.5. Questionnaire – Obtaining Feedback from Students

Name:

Section: Date:

SeamCAD Validation QUESTIONNAIRE

Organizational level

1. How do you rate the way SeamCAD manages organizational levels?
how the model and diagrams can be browsed by going back and forth between, for example, value network, company level and IT level

a. Excellent
b. Good
c. Bad
d. very bad

Any suggestion?

2. Do you think that the top-down approach of SeamCAD has some value in your practice? Please explain in details

a. Yes b. No

Explanation:

Functional level

3. How do you rate the way SeamCAD manages functional levels?
the level of granularity of behavior in a diagram can be changed easily, for example, from "sale" action to three smaller actions "get order", "pay" and "deliver"

a. excellent
b. good
c. bad
d. very bad

Any suggestion?

Multi-system

4. How do you rate the way SeamCAD handles multiple system representation?

    a. excellent
    b. good
    c. bad
    d. very bad

Any suggestion?

Model coherence and Diagrams

5. How do you rate the feature of SeamCAD through which diagrams can be opened as partial views of a common model?

    A diagram in SeamCAD can be opened by choosing a specific object as the context object.
    The context object has the outermost pictogram in the diagram.

    a. excellent
    b. good
    c. bad
    d. very bad

Any suggestion?

6. How do you rate the way diagrams can be customized by hiding / showing specific systems and actions in SeamCAD?

    a. excellent
    b. good
    c. bad
    d. very bad

Any suggestion?

Notation

7. How do you rate the intuitiveness of the notation scheme used in SeamCAD?

    a. excellent
    b. good
    c. bad
    d. very bad

Any suggestion?

Business processes and data flow

8. How do you rate the way business processes are described in SeamCAD?

   a. excellent
   b. good
   c. bad
   d. very bad

   Any suggestion?

9. How do you rate the way data flow is captured in SeamCAD?

   a. excellent
   b. good
   c. bad
   d. very bad

   Any suggestion?

Pedagogical issues

10. What is the difficulty you have in building your models with SeamCAD?

11. In what aspects SeamCAD should be improved to be used as a teaching tool in ESOA course?

12. Your other comments and/or feelings

Appendix C: Modeling Tools in the Fields Related to Enterprise Architecture

Today, there exist quite a large number of modeling tools and generic modeling frameworks. The modeling tools can be roughly categorized into two main groups: software modeling and enterprise modeling. The former aims at providing UML diagrams and some functions to automate the development process (e.g. reverse engineering, code generation, report generation…). The latter provides the modeler with some extra diagrams (may not be UML-compatible) for modeling business processes, organizational units, etc... There are also generic modeling frameworks that allow modelers to quickly define a domain-specific modeling tool.

Rational Software[17], Visual UML[18], UML Studio[19], UML Suite[20], Poseidon[21], Objecteering UML Modeler[22], Microsoft Visio[23] with UML template etc… can be considered as software modeling tools. They support a wide range of UML diagrams that are generally organized into folders or views. These folders and views are typically originated from UML taxonomy on diagrams such as static structure, use-case, implementation etc. This taxonomy is unfortunately not suitable for the representation of the hierarchy of organizational and functional levels. To model a hierarchical system with these tools, the modeler builds several diagrams with the assumption that each of them corresponds to an organizational level. As a consequence, the modeler sees neither the hierarchy of the organizational level nor the traceability between diagrams. In short, we find that the modeler cannot effectively navigate her hierarchical models with these tools.

Enterprise Architect[24], System Architect, Mega[25], Arc Styler, etc… can be considered as enterprise modeling tools. They either provide extra modeling diagrams (beyond UML) or allow the modeler to customize UML diagrams. For example, with Enterprise Architect it is possible to draw any UML element in a specific diagram. The modeler can use UML collaborations, UML actors and UML classes to represent business systems and people collaborating together. However, these tools are still diagram-based. The same comments about model navigation which we made about software modeling tools also apply to enterprise modeling tools.

OpCat [14], the tool for OPM, is more suitable for modeling hierarchical systems because it supports zoom-in/zoom-out operations. In addition, OpCat is a model-based tool. Its diagrams can be created on-demand when the user zooms-in to a process or an object. However, since OpCat does not natively address hierarchical systems, its navigation panel is not used for browsing the hierarchy. It lists diagrams instead.

[17] IBM Rational Software, http://www-306.ibm.com/software/rational/

[18] Visual UML, http://www.visualobject.com/

[19] UML Studio, http://www.pragsoft.com/

[20] UML Suite, http://www.telelogic.com/

[21] Poseidon, http://www.gentleware.com/

[22] Objecteering UML Modeler, http://www.objecteering.com/

[23] Microsoft, Microsoft Visio, http://www.microsoft.com

[24] Enterprise Architect, http://www.sparxsystems.com.au

[25] Mega, http://www.mega.com/

MetaEdit+[26] is considered as a generic modeling tool. The basic rationale behind MetaEdit+ is, at the meta-level, most of modeling tools essentially defines different kinds of objects having some properties and relationships between them. Its main advantage is the ability to quickly define a tool for a given modeling language. Nevertheless, in the aspect as a generalized diagram-based modeling tools, MetaEdit+ also shares the shortcomings with software modeling tools regarding hierarchical systems analyzed above.

GEF[27] allows developers to create a graphical editor for an existing application model. This framework can be used on top of EMF, another framework for data storage, to build a particular modeling tool for hierarchical systems. The main drawback is that the tool built in this way can only be executed within Eclipse and apparently requires quite heavy programming burden. Additionally, the tool graphical pictogram must depend on 2D engineering of GEF, which does not natively support nested notation.

GME is a configurable tool suite that facilitates domain-specific modeling [45]. In GME meta-model, the concept Model can contain other Models, allowing the modeler to establish containment hierarchy in her project. We notice that the tree-view navigation and the way of generating modeling diagrams in SeamCAD are similar to those in GME. The main difference lies in the fact that our tool specifically addresses hierarchical systems in EA by having two model containment hierarchies (functional and organizational) whereas GME was motivated from control systems and integrated circuits (notation is not nested, lack of collaboration modeling).

[26] MetaCase, MetaEdit+, http://www.metacase.com

[27] Eclipse Modeling Framework, http://www.eclipse.org/emf

Appendix D: Design of SeamCAD Tool

This appendix presents the design of the SeamCAD Tool. The tool has client-server architecture. The server side is responsible for model storage and retrieval. The client side offers the modeler with interactive user-interface to edit and to browse her model enterprise. The model is marshaled in Extendible Markup Language (XML)[28] data which is exchanged between the server and the client following the Hypertext Transfer Protocol[29].

Both the server side and the client side were implemented in Java. Figure 69 is a UML component diagram that illustrates how the server side and the client side of the SeamCAD tool communicate with each other. The server side consists of Java servlets[30] that access a MySQL[31] database through Java Database Connectivity[32]. They are deployed in a Tomcat[33] web server. When the modeler opens an existing model, a Java servlet is invoked to load all model elements that belong to this model from the database. These elements are marshaled into an XML document that is immediately sent to the client side. When the modeler saves the model she is editing, the client side marshals the model elements that were modified into an XML document that is immediately sent to the server side. Both the server and the client rely on the same parser to convert the XML document they receive into Java objects that represent the model elements. They ways that model elements are marshaled at the server side and the client side are identical.

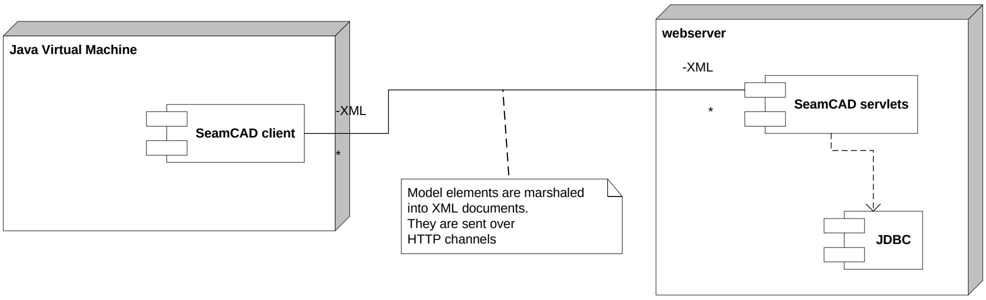

Java Virtual Machine
SeamCAD client
-XML
*
webserver
-XML
*
SeamCAD servlets
JDBC
Model elements are marshaled into XML documents. They are sent over HTTP channels

Figure 69. The Java packages of the implementation of the client side

It is obvious that the implementation of the client side is more immense than that of the sever side because the client deals with model creation, modification, deletion and especially the user-interface. The metrics of the Java code that is presented in Section 4.4 of Chapter 4 is actually about the implementation of the client side. The design of the client follows the Model Controller View (MVC) approach [30]. Figure 70 is another UML diagram that illustrates the main packages of the implementation of the client side and the dependency between them. The packages `dcm` contains classes that describe the building blocks of the SeamCAD modeling language. This package plays the role of the Model in the MVC approach. The package `view` provides classes that can render model elements. The package `action` provides the classes necessary to create

[28] W3C - Extendible Markup Language, http://www.w3.org/XML/

[29] W3C - Protocols, http://www.w3.org/Protocols/

[30] Sun - Java Servlet Technology, http://java.sun.com/products/servlet/

[31] My SQL – open source database, http://www.mysql.com/

[32] Sun - Java Database Connectivity, http://java.sun.com/javase/technologies/database/

[33] Apache Tomcat, http://tomcat.apache.org/

user commands that are then executed upon a user invocation. The package `gui` provides classes for implementing all the Swing[34] windows and dialogs and their accessories. This package, together with the package `view`, plays the role of the View in the MVC approach. The package `seamcad` contains Java classes that control the lifecycle of the Swing windows and implement a façade to the whole data structure of the SeamCAD tool. This package, together with the package `action`, plays the role of the Controller in the MVC approach. Note that there are also sub packages of which names have dot characters according to the package hierarchy.

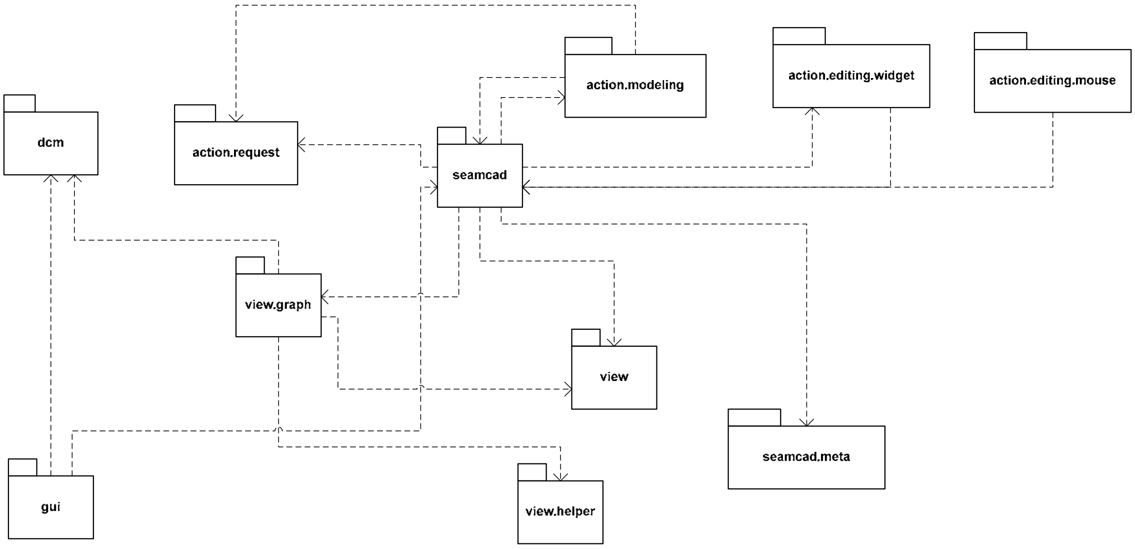


Figure 70. The Java packages of the implementation of the client side

The entire Java code of the client side is documented using JavaDoc[35] and is published on the website of the SeamCAD tool[36].

[34] Sun - Java Swing, http://java.sun.com/docs/books/tutorial/uiswing/

[35] Sun - JavaDoc http://java.sun.com/j2se/javadoc/

[36] SeamCAD Documentation, http://lamspeople.epfl.ch/lsle/SEAMtool/doc/

Bibliographic References

[1] Schekkerman, J., *How to Survive in the Jungle of Enterprise Architecture Framework: Creating or Choosing an Enterprise Architecture Framework*: Trafford, 2004, isbn 141201607-X

[2] Checkland, P. and Scholes, J., *Soft System Methodology in Action*: Chichester UK: Wiley, 1990, isbn 0-471-92768-6

[3] Wegmann., A, Regev, Gil, DelaCruz, Diego, J., Lê, L.-S., Rychkova, and I. Business-IT Alignment with SEAM for Enterprise Architecture. In *Proc. 11th EDOC Conference - The Enterprise Computing Conference*, pp 111-121, IEEE Computer Society, Annapolis, USA, 2007.

[4] Wegmann, A. On the Systemic Enterprise Architecture Methodology (SEAM). In *Proc. 5th International Conference on Enterprise Information Systems* pp 483-49, Angers, France, 2003.

[5] Wegmann, A., Balabko, P., Lê, L. S., Regev, G., and Rychkova, I. A Method and Tool for Business-IT Alignment in Enterprise Architecture. In *Proc. 17th Conference on Advanced Information Systems Engineering Forum*, pp 113-118, FEUP Edições, Porto, Portugal, 2005.

[6] Hevner, S., March, Park, J., and Ram, S., "Design Science Research in Information Systems," *Management Information Systems Quarterly*, vol. 28, pp. 75-105, 2004.

[7] Miller, J. G., *Living Systems*: University of Colorado Press, 1995, isbn 0070420157

[8] OMG, "ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation, X.901, X.902, X.903, X.904, Reference Model of Open Distributed Processing," 1995-1996.

[9] D.Jackson, "Alloy: A lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 256-290, 2002.

[10] Glinz, M., Berner, S., and Joos, S., "Object-oriented modeling with ADORA," *Information Systems - ELSEVIER*, pp. 425-444, 2002.

[11] D'souza, D. F. and Wills, A. C., *Object, Components and Frameworks with UML, The Catalysis Approach*: Addison-Wesley, 1999, isbn 0-201-31012-0

[12] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., and Zettel, J., *Component-based Product Line Engineering with UML*: Addison-Wesley Professional, 2002, isbn 0 201 73791 4

[13] Dori, D., *Object-Process Methodology, A Holistic Systems Paradigm*: Springer Verlag, 2002, isbn 3540654712

[14] Dori, D., Reinhartz-Beger, I., and Sturm, A. OPCAT - A Bimodal CASE Tool for Object-Process Based System Development. In *Proc. 5th ICEIS*, pp Angers, France, 2003.

[15] Lankhorst, M., *Modelling, Communication and Analysis*: Springer, 2005, isbn 978-3-540-24371-7

[16] Dietz, J., *Enterprise Ontology: Theory and Methodology*: Springer, 2006, isbn 3-540-29169-5

[17] Zachman, J. A., "A Framework for Information System Architecture," *IBM System Journal*, vol. 26, pp. 276-292, 1987.

[18] Xia, Y. and Glinz, M. Rigorous EBNF-based Definition for a Graphic Modeling Language. In *Proc. 10th Asia-Pacific Software Engineering Conference*, pp 186-196, IEEE Computer Society Press, Chiangmai, Thailand, 2003.

[19] Atkinson, C., Paech, B., Reinhold, J., and Sander, T. Developing and applying component-based model-driven architectures in KobrA. In *Proc. 5th International EDOC Conference*, pp 212-223, IEEE, Seattle, USA, 2001.

[20] Putnam, J. R., *Architecting with RM-ODP*: Prentice-Hall, 2000, isbn 0-13-019116-7

[21] Lê, L. S. and Wegmann, A. Definition of an Object-Oriented Modeling Language for Enterprise Architecture. In *Proc. 38th Hawaii International Conference on System Sciences*, pp 222a-222a, IEEE, Hawaii, USA, 2005.

[22] Wegmann, A., Lê, L. S., Regev, G., and Wood, B., "Enterprise Modeling Using the Foundation Concepts of the RM-ODP ISO/ITU Standard," *Information Systems and e-Business Management (ISeB) Special Issue on Enterprise Architecture*, vol. 5, pp. 397-413, 2007.

[23] Lê, L. S. and Wegmann, A. An RM-ODP Based Ontology and a CAD Tool for Modeling Hierarchical Systems in Enterprise Architecture. In *Proc. Workshop on ODP for Enterprise Computing, in conjunction with 9th EDOC*, pp 7-15, IEEE, Enschede, The Netherlands, 2005.

[24] Bernardeschi, C., Dustzadeh, J., Fantechi, A., Najm, E., Nimour, A., and Olsen, F. Transformation and Consistent Semantics for ODP Viewpoints. In *Proc. FMOODS'97*, pp Canterbery, UK, 1997.

[25] Lê, L. S. and Wegmann, A., "Meta-model for Object-Oriented Hierarchical Systems," School of Computer and Communication Sciences, EPFL, Lausanne May 2004

[26] Warmer, J. and Kleppe, A., *The Object Constraint Language: Precise Modeling With Uml* Addison-Wesley Professional, 1998, isbn 0201379406

[27] Audi, R., *The Cambridge Dictionary of Philosophy*: Cambridge University Press, 1999, isbn

[28] Lê, L. S. and Wegmann, A. SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture. In *Proc. 39th Hawaii International Conference on System Sciences*, pp 179c-179c, IEEE, Hawaii, USA, 2006.

[29] Lê, L. S. and Wegmann, A., "SeamCAD 1.x: User's Guide," School of Computer and Communication Sciences, EPFL, Lausanne November 2004

[30] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns*: Addison-Wesley, 1995, isbn 0-201-63361-2

[31] Kneitz, E.-M., *Automatic Layout for a Systemic CAD Tool*, Master thesis, in School of Computer and Communication Sciences, EPFL, 2006

[32] Eades, P., Lai, W., Misue, K., and Sugiyama, K., "Layout Adjustment and the Mental Map," *Journal of Visual Languages and Computing*, vol. 6, pp. 183-210, 1995.

[33] Wegmann, A., Lê, L.-S., L.Hussami, and Beyer, D. A Tool for Verified Design using Alloy for Specification and CrocoPat for Verification. In *Proc. First Alloy Workshop, colocated with 14th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp 58, ACM, Portland, USA, 2006.

[34] Beyer, D. Relational Programming with CrocoPat. In *Proc. 28th International Conference on Software Engineering*, pp 807-810, ACM, Shanghai, China, 2006.

[35] D.Beyer and Noack, A., "CrocoPat 2.1 Introduction and Reference Manual," Computer Science Division (EECS), University of California, Berkeley

[36] Wegmann, A., Regev, G., delaCruz, J. D., Lê, L. S., and Rychkova, I., "Teaching Enterprise Architecture in Practice," *Journal Enterprise Architecture*, vol. 3, pp. 15-24, 2007.

[37] Dan, D., *ERP Handbook, Outil d'organisation pour l'intégration et le développement de la solution ERP DOPG Prod.com répondant aux besoins actuels et futurs des clients de DOP Gestion SA*, Master thesis, in School of Computer and Communication Sciences, EPFL, 2008

[38] Regev, G. and Wegmann, A. Where do Goals Come From: the Underlying Principles of Goal-Oriented Requirements Engineering. In *Proc. 13th International Requirements Engineering Conference*, pp 353 - 362, IEEE, Paris, 2005.

[39] Langenberg, K., *Designing Enterprise Architectures with the SEAM Method - In-Depth Study, Application and Critical Analysis*, Master thesis, in School of Computer and Communication Sciences, EPFL, 2004

[40] Kornfilt, M., "SimSeam: Adding Dynamic Simulation To A Graphical, Uml-Like, Modeling CAD Tool," School of Computer and Communication Sciences, EPFL, Lausanne, *Semester project*, June 2005

[41] Forrester, J. W., *Industrial Dynamics*. Cambridge: The M.I.T. Press, 1961, isbn 1883823366

[42] Sood, K., "Interdisciplinary Learnings from a Cross Study of SEAM and System Dynamics," School of Computer and Communication Sciences, EPFL, Lausanne, *Semester project*, September 2004

[43] Kloppmann, M., Koenig, D., Leymann, F., Pfau, G., Rickayzen, A., Riegen, C., Schmidt, P., and Trickovic, I., "WS-BPEL Extension for People – BPEL4People," IBM and SAP, *White paper*, June 2005

[44] Kelly, S. and Tolvanen, J.-P., *Domain-Specific Modeling: Enabling Full Code Generation*: Wiley-IEEE Computer Society Press, 2008, isbn 978-0-470-03666-2

[45] Karsai, G., Maroti, M., Ledeczi, A., Gray, J., and Sztipanovits, J., "Composition and cloning in modeling and meta-modeling," *IEEE Transactions on Control Systems Technology*, vol. 12, pp. 263-278, 2004.

Curriculum Vitae

Mr. Lam-Son LÊ

First name: Lam-Son	Last name: Lê
Nationality: Vietnam	Gender: Male
Date of birth: 7th September 1975	Nationality: Vietnam
Place of birth: Phu Tho Province, Vietnam	Marital status: Married

Education	11/2003 – 11/2008	Doctoral program of I&C, EPFL, Switzerland Dr.Sc (Ph.D.) in Computer and Communication Sciences
	10/2002 – 07/2003	Doctoral school of I&C, EPFL, Switzerland Graduate Certificate
	1993 – 1998	HCMC University of Technology, Vietnam Engineer Diploma on Information Technology
	1990 – 1993	Ly Tu Trong senior high school Can Tho City, Vietnam (specialized in Math)
	1986 – 1990	Doan Thi Diem junior high school Can Tho City, Vietnam (specialized in Math)
Languages	English: fluent in writing and speaking French: able to handle daily communications Vietnamese: native language	
Employment	11/2003 – 12/2008	I&C School, EPFL Lausanne, Switzerland Worked as a research assistant at the Laboratory of Systemic Modeling
	10/2001 – 07/2003	I&C School, EPFL Lausanne, Switzerland Did an internship at the Laboratory of Systemic Modeling
	04/1998 – 06/2001	HCMC University of Technology, Vietnam Worked as an assistant lecturer / researcher at the Department of Information Technology

Awards	In March 2005, awarded a second prize in the Vietnam's best scientific and technological innovations (VIFOTEC) for the project “Management and visualization of the geophysical data of Bach Ho oil field” In May 2004, awarded a first prize in the Contest of Creativity in Science and Technology of HoChiMinh City for the project “Management and visualization of the geophysical data of Bach Ho oil field” In July 2001, granted a 1-year scholarship by FCS (Federal Commission for Scholarships for foreign students) to study in the EPFL, Switzerland. This scholarship was then extended to cover the successive academic year. During the period 1993-1998, had been granted university scholarships that were exclusively given to top undergraduate students. In May 1993, won an incentive prize in Vietnamese Annual Mathematical National Contest for senior high school students. Had also been a candidate in a team-selecting contest of Vietnam for participating in the International Mathematical Olympiad 1993.
Given Presentations	• “An Example of a Hierarchical System Model using SEAM and its Formalization in Alloy” presented at *4th International Workshop on ODP for Enterprise Computing, in conjunction with 11th EDOC*, Annapolis, USA, October 2007 • “From Business to IT with SEAM: J2EE Pet Store Example”, presented at *11th International EDOC Conference - The Enterprise Computing Conference*, Annapolis, USA, October 2007 • “Business-IT Alignment with SEAM for Enterprise Architecture”, presented at *11th International EDOC Conference - The Enterprise Computing Conference*, Annapolis, USA, October 2007 • "SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture", presented at *39th Hawaii International Conference on System Sciences*, Hawaii, USA, January 2006. • “An RM-ODP based Ontology and a CAD Tool for Modeling Hierarchical System in Enterprise Architecture”, presented at *Workshop on ODP for Enterprise Computing, in conjunction with 9th EDOC*, Enschede, The Netherlands, September 2005 • “Solving Delaunay Triangulation Problem on Multiprocessing Environment”, presented at *RESCCE’ 2000*, Ho Chi Minh City, Vietnam, June 2000
Hobbies	Photography, soccer, hiking, sightseeing, traveling, swimming, skiing, scenic drive, windsurfing…

Representative publications

- L. S. Lê and A. Wegmann
  Definition of an Object-Oriented Modeling Language for Enterprise Architecture
  Proceedings of 38th Hawaii International Conference on System Sciences, p. 222a, Track 8, IEEE Computer Society, Hawaii, USA, January 2005.
- L. S. Lê and A. Wegmann
  An RM-ODP Based Ontology and a CAD Tool for Modeling Hierarchical Systems in Enterprise Architecture
  2nd International Workshop on ODP for Enterprise Computing, in conjunction with 9th EDOC, pp. 7-15, ISBN 84-689-3693-6, Enschede, The Netherlands, September 2005.
- L. S. Lê and A. Wegmann
  SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture
  Proceedings of 39th Hawaii International Conference on System Sciences, p. 179c, Track 8, IEEE Computer Society, Hawaii, USA, January 2006.
- A.Wegmann, L. S. Lê, L.Hussami and D. Beyer
  A Tool for Verified Design using Alloy for Specification and CrocoPat for Verification
  First Alloy Workshop, ACM SIGSOFT Sofware Engineering Notes, Volume 31, Number 6, page 42, Portland, Oregon USA, November 2006.
- A.Wegmann, L. S. Lê, G. Regev and B. Wood
  Enterprise Modeling Using the Foundation Concepts of the RM-ODP ISO/ITU Standard
  Information Systems and e-Business Management (ISeB), vol. 5, pp. 397-413, ISSN 1617-9846, Springer Berlin / Heidelberg
- A. Wegmann, L. S. Lê, J. D. De La Cruz, I. Rychkova and G. Regev
  An Example of a Hierarchical System Model using SEAM and its Formalization in Alloy
  4th International Workshop on ODP for Enterprise Computing, in conjunction with 11th EDOC, pp. 21-29, Annapolis, USA, October 2007.

Publication list

Journal Articles (2)

- A.Wegmann, L. S. Lê, G. Regev and B. Wood
  Enterprise Modeling Using the Foundation Concepts of the RM-ODP ISO/ITU Standard
  Information Systems and e-Business Management (ISeB), vol. 5, pp. 397-413, ISSN 1617-9846, Springer Berlin / Heidelberg
- A.Wegmann, G. Regev, J. D. De La Cruz, L. S. Lê and I. Rychkova
  Teaching Enterprise and Service-Oriented Architecture in Practice
  Journal Enterprise Architecture, volume 3, number 4, pp. 15-24

Conference/Workshop Papers (11)

- A. Wegmann, L. S. Lê, J. D. De La Cruz, I. Rychkova and G. Regev
  An Example of a Hierarchical System Model using SEAM and its Formalization in Alloy
  4th International Workshop on ODP for Enterprise Computing, in conjunction with 11th EDOC, pp. 21-29, Annapolis, USA, October 2007.
- I. Rychkova, G. Regev, L. S. Lê and A.Wegmann
  From Business to IT with SEAM: J2EE Pet Store Example
  (short paper) 11th International EDOC Conference - The Enterprise Computing Conference, pp. 495-502, IEEE Computer Society, Annapolis, USA, October 2007.
- A.Wegmann, G. Regev, I. Rychkova, L. S. Lê, J. D. De La Cruz and P. Julia
  Business-IT Alignment with SEAM for Enterprise Architecture
  (regular paper) 11th International EDOC Conference - The Enterprise Computing Conference, pp. 111-121, IEEE Computer Society, Annapolis, USA, October 2007.
- A.Wegmann, G. Regev, J. D. De La Cruz, L. S. Lê and I. Rychkova
  Teaching Enterprise Architecture in Practice
  Trends in Enterprise Architecture Research Workshop, in conjunction with 15th ECIS, Via Nova Architectura, St. Gallen, Switzerland, June 2007.
- A.Wegmann, L. S. Lê, L.Hussami and D. Beyer
  A Tool for Verified Design using Alloy for Specification and CrocoPat for Verification
  First Alloy Workshop, ACM SIGSOFT Sofware Engineering Notes, Volume 31, Number 6, page 42, Portland, Oregon USA, November 2006.
- J. D. De La Cruz, L. S. Lê and A. Wegmann
  Validation of Visual Contracts for Services
  4th Workshop on Modeling, Simulation, Verification and Validation of Enterprise Information Systems, in conjunction with 8th ICEIS, pp. 147-156, INSTICC Press, Paphos, Cyprus, May 2006.

- J. D. De La Cruz, L. S. Lê and A. Wegmann
  VISUAL CONTRACTS - A way to reason about states and cardinalities in IT system specifications
  Proceedings of 8th International Conference on Enterprise Information Systems, pp. 298-303, INSTICC Press, Paphos, Cyprus, May 2006.
- L. S. Lê and A. Wegmann
  SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture
  Proceedings of 39th Hawaii International Conference on System Sciences, p. 179c, Track 8, IEEE Computer Society, Hawaii, USA, January 2006.
- L. S. Lê and A. Wegmann
  An RM-ODP Based Ontology and a CAD Tool for Modeling Hierarchical Systems in Enterprise Architecture
  2nd International Workshop on ODP for Enterprise Computing, in conjunction with 9th EDOC, pp. 7-15, ISBN 84-689-3693-6, Enschede, The Netherlands, September 2005.
- A.Wegmann, P. Balabko, L. S. Lê, G. Regev and I. Rychkova
  A Method and Tool for Business-IT Alignment in Enterprise Architecture
  Proceedings of 17th Conference on Advanced Information Systems Engineering Forum, pp. 113-118, FEUP Edições, ISBN 972-752-078-2, Porto, Portugal, June 2005.
- L. S. Lê and A. Wegmann
  Definition of an Object-Oriented Modeling Language for Enterprise Architecture
  Proceedings of 38th Hawaii International Conference on System Sciences, p. 222a, Track 8, IEEE Computer Society, Hawaii, USA, January 2005.

Technical Reports (2)

- L. S. Lê and A. Wegmann
  SeamCAD 1.x: User's Guide
  Technical report No. IC/2004/98, École Polytechnique Fédérale de Lausanne (EPFL), November 2004.
- L. S. Lê and A. Wegmann
  Meta-model for Object-Oriented Hierarchical Systems
  Technical report No. IC/2004/47, École Polytechnique Fédérale de Lausanne (EPFL), May 2004.