Support for Structuring Mechanism in the Integration of Organizational Requirements and Object Oriented Modeling

Fernanda M. R. Alencar[1], Gilberto A. A. Cysneiros Filho[2], Jaelson F. B. Castro[2]

[1]Universidade Federal de Pernambuco, Departamento de Eletrônica e Sistemas, Recife, Brazil
fmra@ufpe.br
[2]Universidade Federal de Pernambuco, Centro de Informática, Recife, Brazil
{gaacf, jbc}@cin.ufpe.br

Abstract. The success of computer applications depends on a good understanding of the organizational environment. Requirement modeling techniques may be used to help to understand the organizational process. In recent years, we have observed a growing influence of the object-orientation paradigm. Unfortunately the dominant technique of object oriented modeling UML (Unified Modeling Language) is ill equipped to represent the organizational requirements. We advocate the use of the i* technique to model organizational requirements in terms of the organizational relationships among the several organizational actors, as well as a means for understanding the rationale for the decision-making. In this paper we discuss some improved guidelines for the integration of early and late requirements specifications. We also present a prototype tool (GOOD - Goal Object Oriented Development) which support the mapping of i* organizational models stored in a Telos repository by the OME tool into UML elements modeled by the Rational Rose tool.

1 Introduction

A good Requirements Engineering effort is paramount to the success of any kind of system. Often, systems fail to properly support the organizations of which they are an integral part. Primary reasons for such failures are the lack of proper understanding of the organization by the developers of the system, also the frequency of organizational changes, which cannot be accommodated by existing systems (or their maintainers).
Requirements capture has been acknowledged as a critical phase of software development, today in all of kind of systems, precisely because it is the phase which deals not only with technical knowledge, but also with organizational, managerial, economic and social issues. The emerging consensus is that a requirement specification should include not only software specifications but also business models and other kinds of information describing the context in which the intended system will function [1] to achieve successful development of high quality products. However, the production of high quality specifications is not easy. Usually the

customers do not exactly know what they want and sometimes the requirements may not reflect the real needs of the customers. It is common for requirements to be incomplete and/or inconsistent.

At the early phase [2] requirements activities are typically informal and address organizational or non-functional requirements. At the late phase requirements activities usually focus on completeness, consistency, and automated verification of requirements.

The Unified Modeling Language [3] is well suited for late-phase requirements capture. It facilitates the production of a requirement document, to be passed on to developers, so that the resulting system would be adequately specified and constrained in a contractual setting. However, UML is ill equipped for early requirements capture because it can not represent how the intended system meets organizational goals, why the system is needed, what alternatives were considered, what the implication of the alternatives are for the various stakeholders, and how the stakeholders' interests and concerns might be addressed. What is required to capture such concerns is a framework that focuses on the description and evaluation of alternatives and their relationship to the organizational objectives behind the software development project [4]. We argue that the *i** framework [2], is well suited for early-phase requirements capture, since it provides for the representation of alternatives, and offers primitive modeling concepts such as those of softgoal and goal.

Hence, our contention is that UML alone is not adequate to deal with all different types of analysis and reasoning that are required during the requirements capture phases. Instead, we advocate the use of two complementary modeling techniques, *i** and *UML*.

The goal of this paper is to improve the mapping rules presented in [13], to cope with structuring mechanisms supported by the i* technique, namely agents, roles and positions. Therefore, we propose rules to treat these sub-units and their relationships. Hence, we present the transition from informal descriptions of actors and theirs sub-units in *i** to precise requirements in p***UML***. This constitutes a conceptualization activity within which a developer might make use of domain knowledge partly expressed in descriptions of the organization, and partly in existing requirements specifications.

This rest of this paper is organized as follows. Section 2 introduces the language used for the early requirements description, namely the *i** technique. In section 3, we provide some means for transforming the sub-units of a complex actor in *i** models into precise specifications in pUML/OCL. Section 4 describes late requirements of the SmartCD taking for base the early requirements captured in the previous section. In Section 5 we introduce tools used to support *i**, UML and the mapping, namely the GOOD tool. Section 6 describes some related work, while section 7 concludes the paper with a summary of its contributions. Throughout the paper, a small CD store example is used.

2 The i* Modeling Framework

The i* technique [2] provide understanding of the organizational environment and goals. The i* offers a modeling framework that focuses on strategic actor relationships. The term actor was used to refer generically to any unit to which intentional dependencies could be ascribed. An intentional actor does not simply carry out activities and produce entities, but has motivations, intents, and rationales behind its actions [2]. An actor is strategic when it is not merely focused on meeting its immediate goal, but is concerned about longer-term implications of its structural relationships with other actors. Usually, when we try to understand an organization, the information captured by standard modeling techniques (DFD, ER, Statechart, etc.) focuses on entities, functions, data flows, states and the like. They are not capable of expressing the reasons (the "why's") of the process (motivations, intentions and rationales). The ontology of i* [2] caters to some of these more advanced concepts. The participants of the organizational setting are actors with intentional properties, such as, goals, beliefs, abilities and compromises. These actors depend upon each other in order to fulfill their objectives and have their tasks performed. We can model the internal structure of an actor grouping and categorizing dependencies as belonging to sub-units of this actor. Therefore, a finer grouping of dependencies would help identify more precisely how one dependency might lead to other dependencies. The i* technique offers two models: The Strategic Dependency (SD) model, and the Strategic Rationale (SR) model.

2.1 The Strategic Dependency Model

This model focuses on the intentional relationships among organizational actors. It consists of a set of nodes and links connecting them, where nodes represent actors and each link indicates a dependency between two actors. The depending actor is called depender, and the actor who is depended upon is called the dependee. Hence, an SD model consists of a network of dependency relationships among various actors, capturing the motivation and the rationale of activities. *i** distinguishes four types of dependencies. Three of these related to existing intentions – *goal dependency*, *resource dependency* and *task dependency*, while the fourth is associated with the notion of non-functional requirements, the so-called *softgoal dependency*. In a *goal dependency,* an agent depends on another to fulfill a goal, without worrying how this goal will be achieved. In a *resource dependency,* an agent depends on another agent to provide a physical resource or information. In a *task dependency,* an agent depends on another to carry out a task. A *softgoal dependency* is similar to a *goal dependency*, except that a softgoal is not precisely defined. In *i** we can also model different degrees of dependency commitment on the part of the relevant actors (open, committed, or critical). To model the sub-units of a complex actor, we can also classify actors into three types of sub-units - *agents*, *roles,* and *positions* – each of which is an actor in more specialized sense.

An *agent* is an actor with concrete physical manifestations (a person or a system). An agent has dependencies that apply regardless of what roles he/ she/it happens to be

playing. These characteristics are not easily transferable to other individuals. The actor has skills, experiences, and its physical limitations. An actor plays a role.

A *role* is an abstract characterization of the behavior of an actor within some specialized context, domain or endeavor. Its characteristics are easily transferable to other actors. Dependencies are associated with a role when these dependencies apply regardless of who plays the role.

A *position* is intermediate in abstraction between a role and an agent. It is a set of roles typically played by one agent. We can say that an agent occupies a position and that a position *covers* a role.

Finally, *i** supports the analysis of opportunities and vulnerabilities for different actors [2].

Suppose a situation in which a Client wishes to buy CDs and goes to a specialized store because its services are of good quality and it claims to have most (if not all) available titles on stock. If a client cannot find his/hers preferred title, the shop can happily place an order for it and notify the client upon its arrival. The shop has decided to improve its services by commissioning a new software system (SmartCD) to handle orders as well as providing an online catalogue (it would be so convenient!). In the figure 1, we have the Strategic Dependency (SD) model of the CD store case study.

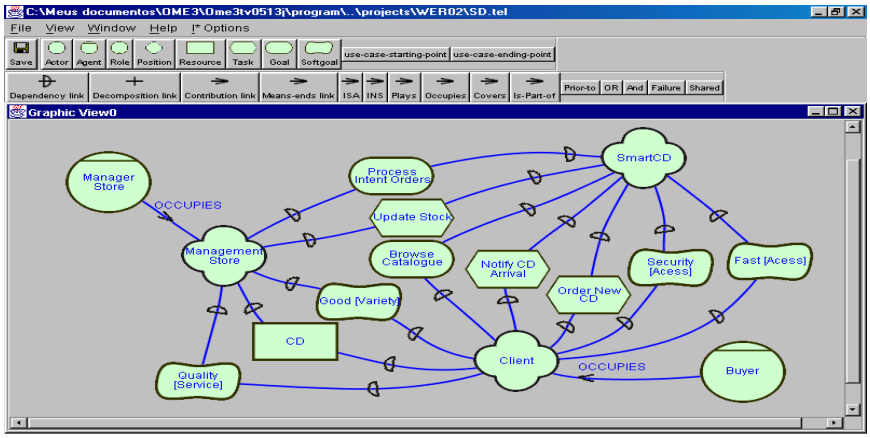

Fig. 1. Strategic Dependency Model

At this early stage of requirements capture we have identified three positions: *Client*, *Management Store* and *SmartCD*. This last actor corresponds to the system to be developed, handling orders, notifications of CD arrivals and providing the online catalogue. The dependencies between the *Client* and the *Management Store* position (actor) can be find in figure1. The *Client* depends on the *Management Store* for getting the CD (resource dependency). However, he/she wishes the services to be of good quality (softgoal *Quality[Service]*) and the store to maintain a good stock of CDs (softgoal *Good variety*). Of course these goals are not yet precisely defined at this early stage, hence the use of *softgoals*. The *Manager Store* agent Turning to the

relationship between actors *Client* and *SmartCD,* we notice that one of the goals for introducing the online system is to enable browsing facilities (goal dependency *Browse Catalogue*). In fact, the store may stock thousands of CDs, making it difficult (or even impossible) for a customer to manually search all of them. In the (unlikely) situation that a CD is not on stock, the *SmartCD* actor will be able to handle orders online (the system will inform what and how it should be done, hence task dependency *Order new CD*). This feature is much awaited, since filling orders manually (through a sales person) is time consuming. Of course, when the (ordered) CD arrives, the *Client* will be notified as soon as possible (actually there is a pre-defined procedure for dealing with it, hence the task dependency *Notify CD Arrival*). Last but not least, the *Management Store* actor also has some expectations on the commissioned *SmartCD* system. It expects the access to be fast (softgoal *Fast[Access]*) and to use it to keep the stock updated (task *Update Stock*).

In figure 2, we concentrated our specification on the *SmartCD* actor. As we can see, we find the five types of relationship – occupies, covers, play, is-Part-of and is-a. The first is respectively between an agent - *SystemControl* - and a position - *SmartCD*. The second one is among a position – *InternetSales* - and a set of roles - *CD_Reserve* and *CD_Delivery*. The third is between an agent – *Office_Boy* - and a role - *CD_Delivery*. Roles, positions, and agents can each have subparts. It is expressed by the fourth relationship "*IS-PART-OF*" construct. Thus, the position *SmartCD* consists of *InternetSales*, *Inventory*, and *Financial*. The fifth relationship, *IS-A* construct represents a conceptual generalization/specialization among agents, positions or roles. This construct is not described in figure 2.

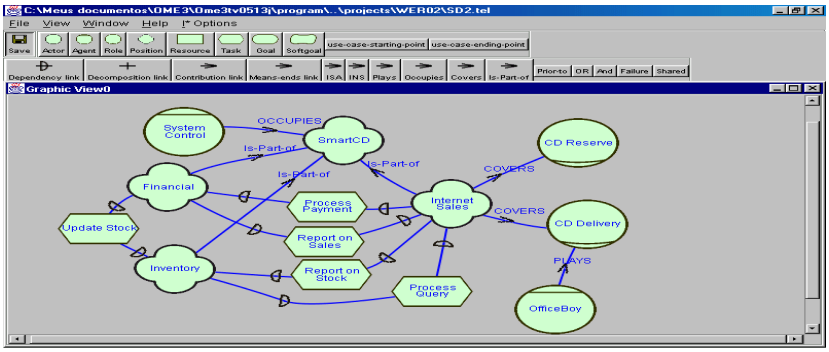

Fig. 2. SmartCD Strategic Dependency Model

There can be dependencies among the agents, positions, and roles. The *Internet Sales* position depends on the *Financial* position to process the payment of the CDs bought by some *Client*. This dependency is expressed by the task dependency "*Process Payment"*. In the same way, the *Internet Sales* agent depends on the *Inventory* agent to have the reports of the availability of products in stock (the task dependency "*Report on Stock*"). The *Financial* position depends on the *Internet Sales* position to inform the data of the sales, for example, *Client*'s data, the purchase value,

payment form, delivery local, amount and type of product sold. This dependency is expressed by the task dependency "*Report on Sales*". The *Financial* position also depends on the agent *Inventory* to inform update information of the available products in the stock (the task dependency "*Update Stock*"). Finally, the *Inventory* position depends on the *Internet Sales* position to request information on the Store's stock of products (the task dependency "*Process Query*").

2.2 The Strategic Rational Model

The Strategic Rationale Model (SR) provides a more detailed level of modeling by looking "inside" actors to model internal intentional relationships. It is used to: (i) describe the interests, concerns and motivations of participants process; (ii) enable the assessment of the possible alternatives in the definition of the process; and (iii) research in more detail the existing reasons behind the dependencies between the various actors. Nodes and links also compose this model. It includes the previous four types of nodes (present in the SD model): *goal*, *task*, *resource* and *soft-goal*. However, two new types of relationship are incorporated: *means-end* that suggests that there could be other means of achieving the objective (alternatives) and *task-decomposition* that describes what should be done in order to perform a certain task.

In figure 3 we use de SR notation to detail the *InternetSales* actor. Due to space limitation we now only comment some aspects. An interested reader can find a fuller description of the approach in [13].

The store is interested in attracting (new and old) clients. In the InternetSales module several strategic decisions were taken in consideration and as a result the task Interact by Site was decomposed into three aspects (expressed by a task-decomposition link):

- To define standard procedures to insert the user data on the system (captured by the sub-task *RegisterClient)*;
- To define standard procedures for assist the Client on his/her necessity;
- To meet the objective of have the CD wanted (captured by the sub-goal *SearchCD*): two alternatives are considered for meeting this objective: to execute the procedures for the fast search of items (sub-task *FastSearch*) or to execute the procedures for super search of items (sub-task *SuperSearch*).

At this point, we may stop the process of modeling the strategic dependencies of the CD store. We are already capable of understanding some issues of the application domain (the enterprise). We can then move to provide a detailed system specification.

3 Mapping Early Requirements into Late Requirements

Late requirements focus on the functional and non-functional requirements of a system-to-be, which will support the chosen alternative among those considered during early requirements. To specify the late requirements, we adopt pUML (precise UML) [5], which provides a precise denotational semantics for core UML elements, such as: relationship, classifier, association, and generalization.

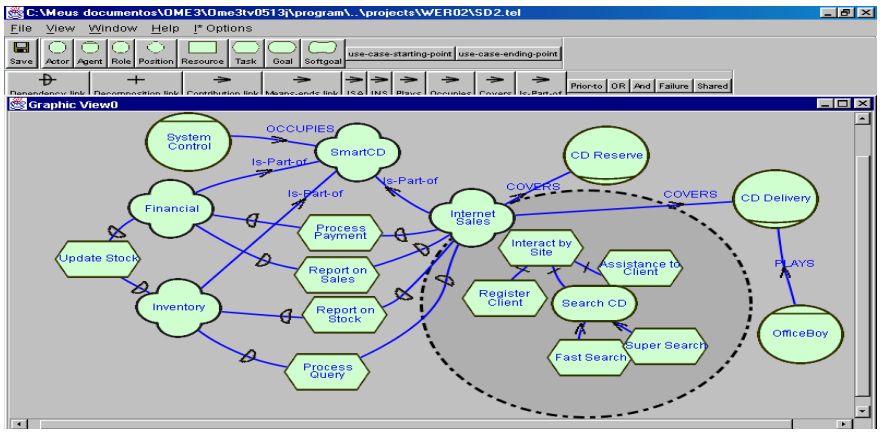

Fig. 3. Strategic Rationale Model of the SmartCD

The pUML diagrams alone are not sufficient for late requirement capture because it does not provide for the specification of constraints, such as invariants, preconditions and the like. For this task, we have adopted the Object Constraint Language (OCL) [6]. OCL is a textual language, also part of the Object Management standard, that can precisely describe constraints for object oriented models.

The transformation of *i**-based early requirements into pUML/OCL-based late requirements is founded on six guidelines which deal respectively with actors, tasks, resources, goals/softgoals, task decompositions and means-ends relationships [13].

A meta model to specify the mapping rules more exactly is being considered in works that are being developed in our group.

The work consist in extend the guideline G1 related with the mapping of i* actors to pUML classes proposed in [13].

The new guidelines will be denoted by G'. Recall from section 2.1 that i* actors can also be classified into three types of sub-units - agents, roles, and positions – each of which is an actor in more specialized sense. We propose a new set of six sub-guidelines that complement the original set described in [13]. From the i* models (figure 1 and 2) and with the enhanced guidelines we are able to construct the class diagram shown in figure 4.

Guideline G'1.1:
i agents or i* roles or i* position can be mapped to pUML classes. OCL constraints can be attached to the actor-generated classes.*

There were eighth actors in our case study (see figure 3): *SmartCD, Financial, Internet Sales* and *Inventory* (positions), *System Control* and *Office Boy* (agents), and *CD Reserve* and *CD Delivery* (roles). These can be mapped to the classes shown in figure 4.

The subparts of roles, positions, and agents are expressed by an "IS-PART-OF" construct as we can see with the position *SmartCD* (see figure 2).

Guideline G'1.2:

The i relationship **IS-PART-OF** between positions, agents or roles can be mapped as a class aggregation in pUML.*

The position SmartCD is composed by the positions Internet Sales, Inventory, and Financial (see figure 3). In pUML (see figure 4), SmartCD class is the aggregate of three corresponding composite classes.

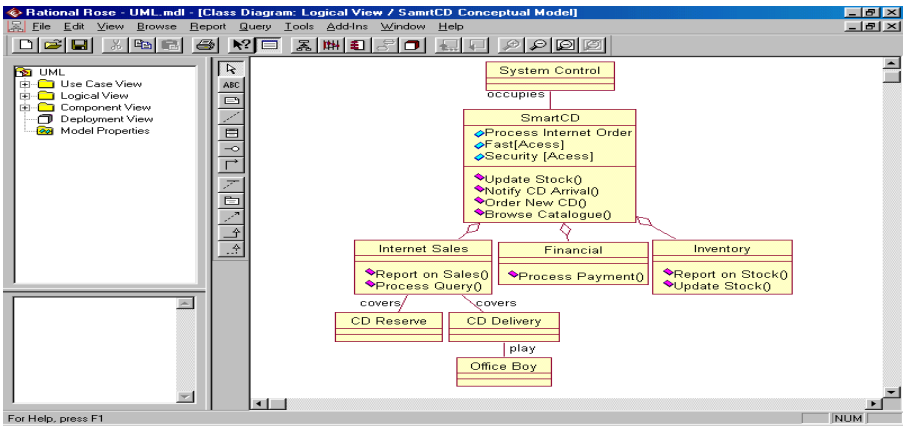

Fig. 4. Context Class Diagram of the SmartCD System

Guideline G'1.3:
The i* relationship **IS-A** between positions, or agents, or roles can be mapped as a class generalization/specialization in pUML.

In our study case we do not have this type of relationship.

Guideline G'1.4:
The i relationship **OCCUPIES** between an agent and a position can be mapped as a class association in pUML named OCCUPIES.*

The agent SystemControl OCCUPIES the position SmartCD (see figure 3). In pUML (see figure 4), there is an association between SystemControl class and SmartCD class.

Guideline G'1.5:
The i relationship **COVERS** between a position and a role can be mapped as a respective class association in pUML named COVERS.*

The position *Internet Sales* ***COVERS*** the role *CD Delivery* and **COVERS** the role *CD Reserve* (see figure 3). In pUML (see figure 4), there is an association between ***Internet Sales*** class and ***CD Delivery*** **class**, and there is an association between ***Internet Sales*** class and ***CD Reserve*** class, which are named COVERS.

Guideline G'1.6:
The i* relationship **PLAYS** between an agent and a role can be mapped as a respective class association in pUML named PLAYS.

The agent *OfficeBoy* ***PLAYS*** the role *CD Delivery* (see figure 3). In pUML (see figure 4), there is an association between ***OfficeBoy*** class and ***CD Delivery*** class, in pUML named ***PLAYS***.

The remaining set of guidelines described in [13] can be used to complete the context diagram presented in figure 4.

Of course not all concepts captured in the early requirements phase will correspond to software system models. The models do not have a one-one relationship; many elements of the organizational model are not part of the software model, since not all of the organizational tasks require a software system. Many tasks contain activities that are performed manually outside the software system, and so do not become part of the software system model. Likewise, many elements in the software model comprise detailed technical software solutions and constructs that are not part of the organizational model. Nonetheless, as we shall see, pUML/OCL also can be used to represent this information.

In this sense having already understood the context of the application with the use of the organizational modeling technique i * we started to specify the functionality of SmartCD in the next session.

4 Some Late Requirements of the SmartCD

Late requirements analysis results in a requirements specification document which describes all functional and non-functional requirements of the system-to-be. In our case study, the *SmartCD* actor represents the system that will help the store attract customers (see figure 2). The system is structured in terms of three sub-units, for *Inventory*, *Financial* and *Internet Sales*.

We concentrate here on the *Internet Sales* subsystem, since it is related to the *sales by web* proposal. We can consider the context class diagram of figure 4 as the starting point of our discussion, which is not intended to be exhaustive. As a matter of fact, methods such as Catalysis [7] already make some of these issues that arise below very clear.

The *Internet Sales* subsystem provides a website which enables customers to remotely access the store. This site allows visitors to search for CDs, also see information about pop stars and musical events.

There are two ways for a client to search for a CD: the *fast* and *super* search. When in *fast* mode, the visitor provides the name of the album, artist or music she is looking for. The super search, on the other hand, is intended to help those who still do not know what they are looking for. During *super* search, the user can (optionally) provide the name of the album, music, styles (Pop, Rock, Rap, Reggae, Jazz, etc.), the recording, the repertoire (national or international) and the release time.

Assistance can be provided upon request by e-mail or through a FAQ page, which contains answers to Frequently Asked Questions. If the visitor does not find an answer, she can fill a form (including the subject, name, and number of the purchase request) and submit it. Upon receipt, an on-line sales assistant will answer the question.

A visitor (if not already a client) would have to register in order to use the system.

Based on these extra descriptions of the system and the initial context class diagram (see Fig. 4), we can now evolve to a new class diagram of the *Internet Sales* as depicted in figure 5.

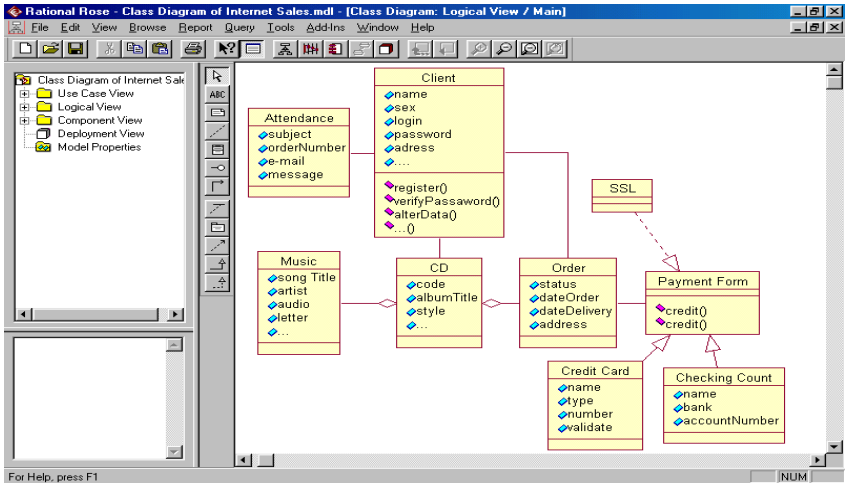

Fig. 5. Context Class Diagram of the Internet Sales Sub-Unit

5 Tool Support

In this section we describe tools that can be used to support modeling in i* and UML. We begin describing the OME toolset. Then we proceed to review some extension mechanisms available in the Rational Rose environment and conclude describe the GOOD tool.

5.1Organizational Modeling Environment – OME

OME is a goal-oriented and/or agent-oriented modeling and analysis tool. OME is being developed at the Knowledge Management Lab at the University of Toronto. The OME tool currently supports the i*, NFR (Non Functional Requirements) and GRL (Goal-oriented Requirement Language) modeling.

The OME tool is mainly composed of two parts: the OME kernel and the Plugins. OME kernel has a layered architecture, comprised of three major modules (View Layer, Model Framework Layer and KB Layer), each of which consists of several Java classes and/or external tools. Plugins are classes implementing framework specific functionalities, which will be coupled with the OME kernel and provide service to the user together at run time. In each execution of the tool, different kinds

of plugins are loaded according the frameworks (*i**,NFR, GRL) that the current model is based on. Besides OME kernel and plugins, there are also a group of files describing the configuration of frameworks.

The KB (Knowledge Base) Layer is responsible to store the objects in a model, their relationships, and their attributes (pertinent to the model). The major module in the KB is a *Telos* [12] repository.

A Telos knowledge base consists of structured objects built out of two kinds of primitive units, individuals and attributes. Individuals are intended to represent entities (such as a Custumer, Bank Cpy, Media, …), while attributes represent binary relationships between entities or between relationships. Each proposition in *Telos* (individuals or attributes) is defined as a quadruple with components *from, label, to, when*. These denote the source, label, destination and duration of the proposition respectively.

Although propositions are organized in *Telos* along simple classes (having only tokens as instances), *metaclasses* (having only simple classes as instances), *metametaclasses* (having only simple classes as instances, *metametaclasses* and so on) for the objective of the construction of the GOOD tool, only tokens were used because they represent concrete entities of a model.

5.2 Extension mechanism for Rational Rose

The Rational Rose is a visual modeling tool that supports Object Oriented Modeling in UML. It is available in multiple platform (UNIX, Windows, Linux), has integration with multiple other tools CASE (such as Rational RequisitePro, Microsoft Project, Borland Jbuilder), support generation of code, reverse engineering and round trip engineering. Besides these features the Rational Rose provides an interface (Rose Extensibility Interface - REI) that makes it possible to customize and extend it.

The REI Model is essentially a Meta model of a Rose model, exposing the packages, classes, properties, and methods that define and control the Rose application and all of its functions. The details on the classes contained in each package, properties and methods of each class can be found in [13] and in the Help online of the tool Rational Rose.

To communicate with the Rose tool we cab write scripts that access the REI model. The Rational Rose Scripting language is an extended version of the Summit Basic Script Language. It allows the automation of Rational Rose-specific functions, and in some cases even the execution of some functions that are not available through the Rational Rose user interface.

5.3 GOOD (Goals into Object Oriented Development) Tool

GOOD is the prototype of a tool that makes the mapping of the descriptions of the organizational requirements modeled in *i** (modeled by the OME tool) in UML Class Diagram (supported the Rational Rose tool).

The mapping is based in the guidelines presented in [13]. Figure 6 illustrates the components used for the mapping: the repository of data of the tool OME, the tool

GOOD and Rational Rose. The tool GOOD is responsible for the mapping of the files stored in the repository of the tool OME that represent organizational models in *i* * for a conceptual model of the system in the tool Rational Rose.

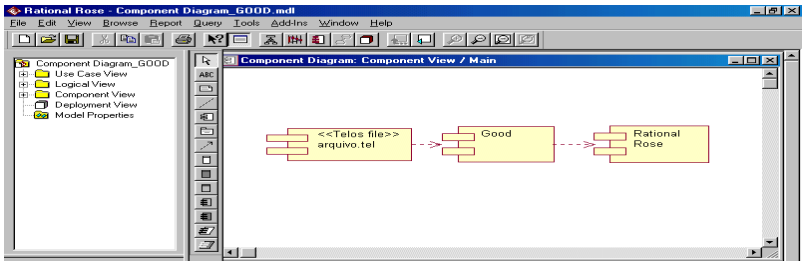

Fig. 6. Component of the Mapping

The GOOD tool can be seen as a Rational Rose extension. It was built using the language Rose Scripting that allows the access to functions of Rose through pre-defined API REI. After installed, the GOOD tool can be invoked through the Rose Tool main menu Tool-> GOOD.

The GOOD tool consists of two components: Configuration Mapping and Execution Mapping. The Configuration Mapping component allows the user of the tool choose the best way (or set of rules) for mapping the organizational models in *i** to UML Class Diagrams. The Configuration Mapping component can be accessed through Rose menu Tool->GOOD-> Configuration Mapping. Figure 7 shows the screen that is exhibited when this option is selected.

The listbox shows the mapping options. These values are derived from the configuration file (mapSetting.cfg). The default setup as in the original rules is presented in [13]. However, these values can be easily modified to add or modify mapping rules. Work is under way to support the new guidelines described in this paper.

For example i* can be mapped to a class or to an actor (stereotype of a class) in UML, a task represented in SD (Strategic Dependency) model can be mapped to an public operation in class that represent the dependee (actor responsible to execute the task) or to a use case (or even to stereotype of a class (<<Task>>)), while a resource represented in SD (Strategic Dependency) i* model can be mapped to a class.

The Execution Mapping component is responsible for executing the mapping. It can be accessed through the Rational Rose menu Tool->GOOD-> Executation Mapping. The first activity that the script execute is to prompt the user to select the path were the organizational i* models are stored (a Telos file). Then the script reads the selected file and captures the i* elements (actors, tasks, resources, goals and softgoals) as well as their relationships (dependency links, means-end links and task decompositions). Next the GOOD tool read a configuration file (mapSetting.cfg, wich stores the mapping rules). Finally it then executes the mapping, creating a class diagram such as the one described by figure 4.

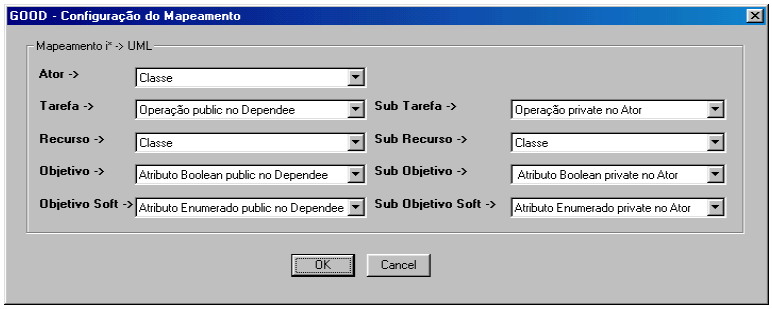

Fig. 7. The Screen of the Configuration Mapping

6 Related Work

The area of Requirements Engineering [7] has developed several novel techniques for early requirements capture [8]. Bubenko emphasizes the need to model organizations and their actors, motivations and reasons [8]. In his work, enterprise modeling and requirements specification are based on the notion that a requirements specification process, from a documentation point of view, implies populating (instantiating) five interrelated sub-model, representing areas of knowledge of the organization, which include an Objectives Model, an Activities & Usage Model, an Actors Model, a Concepts Model, and an Information Systems Requirements Model. Since the models are informal, or at best semi-formal, only some verification can be performed automatically, such as syntactical correctness and connectedness.

In the KAOS framework [8] goals are explicitly modeled and simplified (reduced) through means-end reasoning until it reaches the agent level of responsibilities. KAOS provides a multi-paradigm specification language and a goal-directed elaboration method. The language combines semantic nets for conceptual modeling of goals, requirements, assumptions, agents, objects and operations in the system; temporal logic for the specification of goals, requirements, assumptions and objects; and state-based specifications for the specification of operations. However, agents are expected to behave as prescribed. This feature makes it difficult to analyze strategic relationships and implications in KAOS.

Another important issue related to early phase requirements capture is the representation of qualities attributes, such as accuracy, performance, security, modifiability, etc. In [9] a comprehensive approach for dealing with non-functional requirements - NFR is presented. Structured graphical facilities are offered for stating NFRs and managing them by refining and inter-relating NFRs, justifying decisions, and determining their impact. A current research topic is the extension of traditional Object-Oriented Analysis to explore the alternatives offered by the non-functional goal-oriented analysis, which systematizes the search for a solution which characterizes early phases or requirements analysis, rationalizes the choice of a

particular solution, a relates design decisions to their origins in organizational and technical objectives [10].

Although UML has been used mainly for modeling software, recent proposals have used it for describing enterprise and business modeling. For example, [1] claims that UML is a suitable language for describing both the structural aspects of business (such as the organization, goal hierarchies, or the structure of the resources), the behavioral aspect of a business (such as the processes), and the business rules that affect structure and behavior. In [11] UML is used, from a business perspective, to describe the four key elements of an enterprise model: purpose, processes, entities and organization. The challenge is to transfer the information available in the (early) business models to the (late) software requirements models.

7 Conclusion

In this paper, we have suggested that requirements capture has to be done at different levels of abstraction (ranging from the early phase to the late phase requirements). Furthermore, we argue that UML alone is not adequate to deal with all different types of analysis and reasoning that are required during the requirements capture phases. Instead, we advocate the use of two complementary modeling techniques, i* and a precise subset of UML.

To model and understand issues of the application and business domain (the enterprise) a developer can use the *i** framework which allows a better description of the organizational relationships among the various agents of a system as well as an understanding of the rationale of the decisions taken. For later requirements capture we suggest the use of pUML, a subset of UML, which has a well-defined semantics. Annotations in OCL can also be deployed for describing constraints on the models. We believe that structuring mechanism present in i* framework, such as agent, role and position are appropriate to describe complex systems. Thus we improved previous guidelines to support their mapping. Furthermore, we believe that each language has its own merits for supporting requirements capture. But as long as different techniques are used, then a key issue is the development of an integrated framework to support and guide the interplay of requirement captures activities at the various levels, and to support traceability and change management. Indeed, the guidelines presented in the paper are important steps in this direction. They can help to map the descriptive, early requirements model of the *i** technique into a prescriptive, late requirements model expressed in pUML/OCL.

Further some real industrial case studies are also expected. Work is underway to provide some tool support for the mapping.

References

[1] Erikson, H. and Penker, M.: “Business Modeling with UML: Business Patterns at Work”. OMG Press .John Wileys & Sons 2000.

[2] Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of IEEE International Symposium on Requirements Engineering - RE97, pp.226-235, Jan. 1997.
[3] Booch, G., Jacobson, I., Rumbaugh, J.: Unified Modeling Language User Guide. (ed.): Rational Software Corporation. Addison-Wesley Object Technology Series. Jan., 1999.
[4] Mylopoulos, J., Chung, L., Yu, E.: From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM, 42(1): 31-37, January 1999.
[5] Evans, A., Kent, S.: Core Meta-Modelling Semantics of UML: The pUML Approach. UML'99 – The Unified Modeling Language. Proceedings of <<UML>>'99 The Unified Modeling Language: Beyond the Standard - The Second International Conference. (eds.): Robert France and Bernhard Rumpe. Fort Collins, CO, USA. pp.140-15. Oct 1999.
[6] Warmer, Jos B., Kleppe, Anneke G.: The Object Constraint Language: Precise Modeling with UML. (ed.): Addison-Wesley Object Technology Series. March, 1999.
[7] D'Souza, D. F., Wills, A. C.: Objects, Components, and Frameworks with UML. The Catalysis Approach. (ed.): Addison-Wesley. 1999.
[8] van Lamsweerde, A., Darimont, R., Letier, E.: Managing Conflicts in Goal-Drivel Requirements Engineering. IEEE Transaction on Software Engineering, Special Issue on Inconsistency Management in Software Development, November 1998.
[9] Chung, L. K., Nixon, B. A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering, Kluwer Publishing, 2000.
[10] Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Extending Object-Oriented Analysis to Explore Alternatives. Submitted for publication.1999.
[11] Marshal, C.: Enterprise Modeling with UML: Designing Successful Software through Business Analysis. (ed.): Addison-Wesley Object Technology Series. 2000.
[12] Mylopoulos, J., Borgida, A., Jarke, M., Jarke, M., Telos: Representing Knowledge About Information Systems, ACM Transactions on Information Systems, October, 1990.
[13] Castro, J., Alencar, F., Cysneiros, G.: Integrating Organizational Requirements and Object Oriented Modeling. In: Fifth International Symposium on Requirements Engineering - RE'01, Toronto. 2001.