New Mechanisms for the Integration of Organizational Requirements and Object Oriented Modeling

Fernanda M. R. Alencar[1], Flávio Pedroza[1], Jaelson F. B. Castro[2] and Ricardo C. O. Amorim[2]

[1] Universidade Federal de Pernambuco, Departamento de Eletrônica e Sistemas, Recife, Brazil
fmra@ufpe.br
flavio.pedroza@caeser.org.br
[2] Universidade Federal de Pernambuco, Centro de Informatica, Recife, Brazil
{jbc,rcoa}@cin.ufpe.br

Abstract. The success of computer applications depends on a good understanding of the organizational environment. Thus, requirement-modeling techniques may be used to help to understand a process in terms of goals, business rules, tasks, resources and the relationship between their actors. We have observed a growing influence of the object-orientation paradigm but the dominant technique of object oriented modeling UML (Unified Modeling Language) is still ill equipped to represent the organizational requirements. So, we have advocated the use of the i* technique to model requirements in terms of the 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 extend the i * technique so that we establish an order in the execution of the i* task dependency. We are also proposing to extend the prototype tool (GOOD - Goal Object Oriented Development) to support the new guidelines.

Keywords: Requirements Engineering, The Integration of Early and Late Requirements, Object Oriented Development, Tool Support.

1. Introduction

Requirements capture has been acknowledged as a critical phase of software development. It 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 specification but also business models and other kinds of information describing the context in which the intended system will function [1]. Usually the customers do not exactly know what they want and sometimes the requirements may not reflect the real needs of the customers. At the early phase [5] 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 [2] 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 implications 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 [3]. We argue that the *i** framework [5], 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. Instead, we advocate the use of two complementary modeling techniques, *i** and *UML*. Thus, we want to keep the consistency between the desired software system and the organization objectives, as well to establish the impact that any change of objectives will be able to cause in the system and vice versa.

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 new 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 ***UML***.

2 The i* Modeling Framework

The *i** technique [5] provides 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 [5]. 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.) are not capable of expressing the reasons (the "why's") of the process (motivations, intentions and rationales). The ontology of *i** [5] 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. The *i** technique [5] offers two models: The Strategic Dependency (SD) model, and the Strategic Rationale (SR) model.

2.1 The Strategic Dependency (SD) 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 related to existing intentions: *goal dependency* (ex.: *Browse Catalogue* in figure 1); *resource dependency* (ex.: *Personal Data* in figure 1); *task dependency* (ex.: *Update Stock in figure* 1).

The fourth is associated with the notion of non-functional requirements, the so-called *softgoal dependency* (ex.: *Security [Access]* in figure 1). 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). Ex.: *Store Manager* in figure 2.
- A *role* is an abstract characterization of the behavior of an actor within some specialized context, domain or endeavor. Ex.: *CD Reservation* in figure 2.
- 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. Ex.: *Store Management* in figure 1.

Suppose a situation in which a Client wishes to buy CDs and goes to a specialized store. 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.

In figure 1, we have the initial Strategic Dependency (SD) model of the CD store case study.

At this early phase of requirements capture we have identified three positions: *Client*, *Store* Management and *SmartCD*. This last actor corresponds to the software system to be developed, handling orders, notifications of CD arrivals and providing the online catalogue. The dependencies between the *Client* and the *Store* Management position (actor) can be found in Figure 1.

In Figure 2, we concentrate our specification on the *SmartCD* position. This is the information system that will be developed in the future. As we can see, we use the five types of relationships – 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_Reservation* 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 *SmartCD* position 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 used in Figure 2.

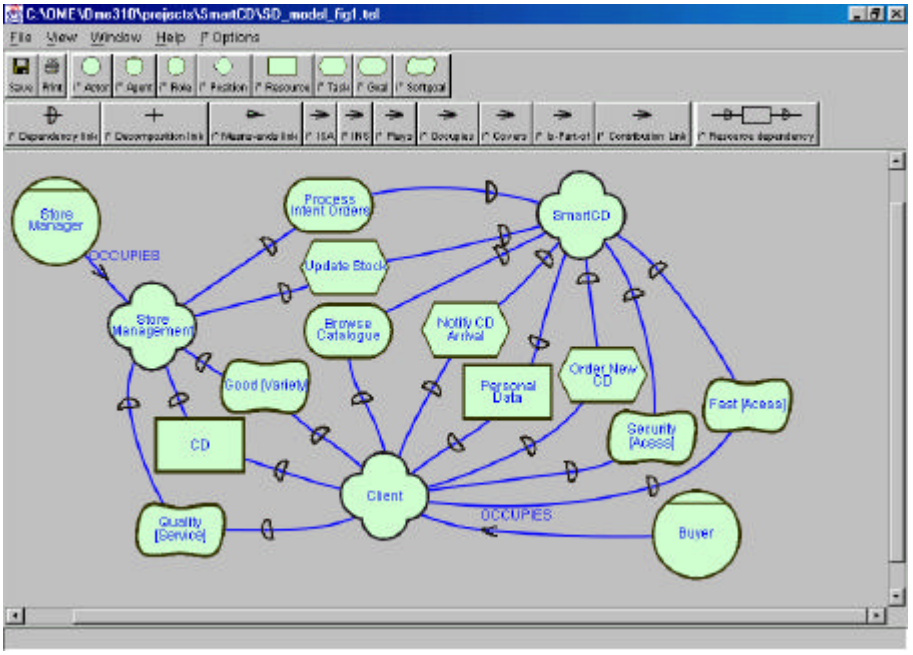

Fig.1. The Strategic Dependency Model

2.2 The Strategic Rational (SR) Model

The SR Model 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. 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. Unfortunately the current Strategic Rationale (SR) model does not capture the order in which the tasks can be decomposed. In order to be capable of capturing the occurrence order of the sub-tasks, we propose the insertion of numeric labels to capture the sequence of occurrence. When the order is not relevant, the elements will be contained in a box of stippled borders. In Figure 3 we use the enhanced Strategic Rationale (SR) notation to

detail the *InternetSales* position, where we will emphasize the order of decomposition of the sub-task.

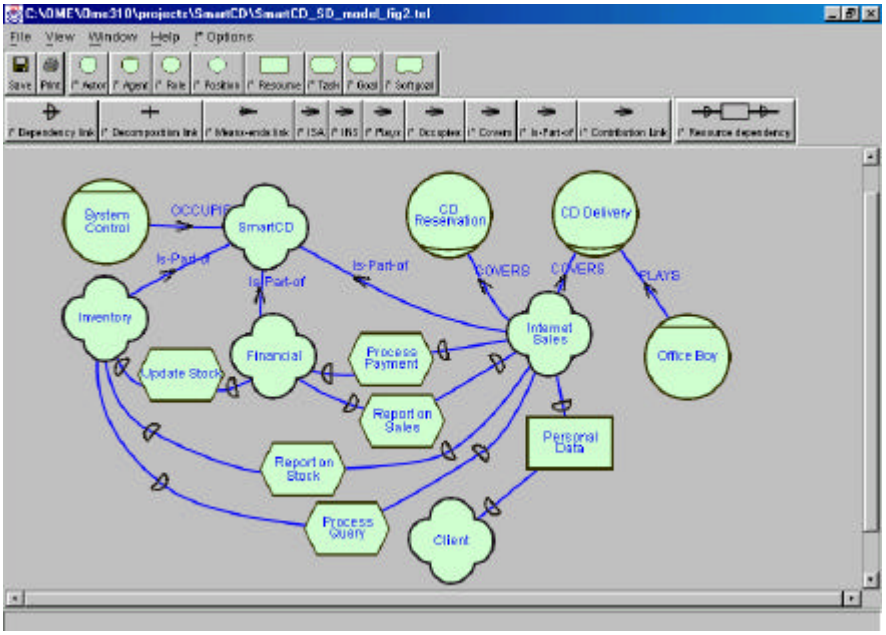

Fig. 2. SmartCD SD Model

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 sub-tasks (expressed by a task-decomposition link) (Figure 3).

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

To specify the late requirements, we adopt pUML (precise UML) [6], which provides a precise denotational semantics for core UML elements (relationship, classifier, association, and generalization).

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) [4]. OCL is a textual language, also part of the Object Management standard, that can precisely describe constraints for object oriented models.

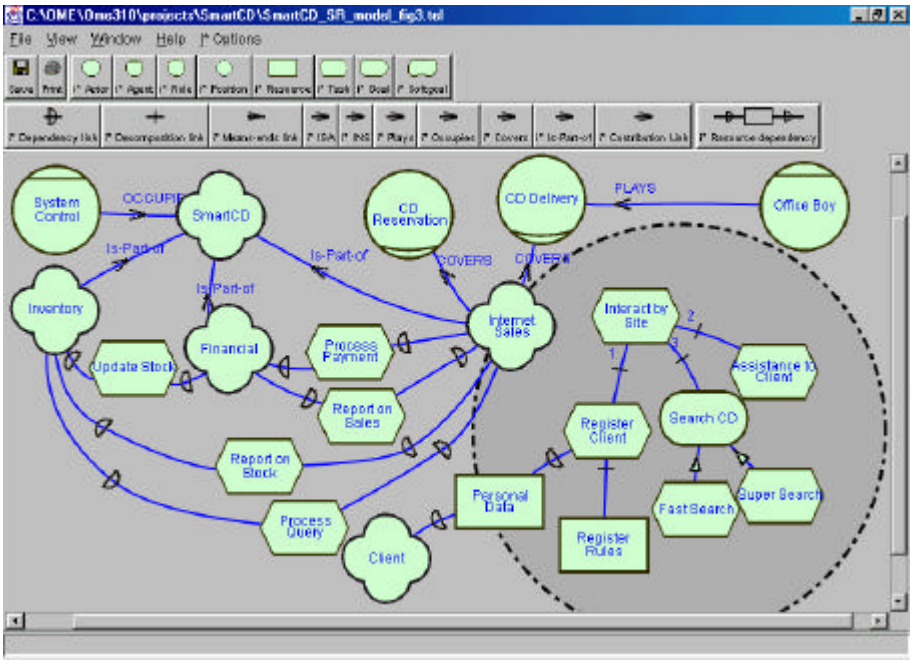

Fig. 3. SR Model of the SmartCD

Original Mapping Guidelines			Extended Mapping Guidelines		
Number	i*	pUML	Number	i*	pUML
G1	Actor	Class	**G'1. 1**	Agents, roles or position.	Class.
		Class	**G'1. 2**	Relationship ***IS-PART-OF*** between positions, agents or roles.	Class aggregation.
		Class	**G'1. 3**	Relationship ***IS-A*** between positions, agents or roles.	Class generalization/specialization.
		Class	**G'1. 4**	Relationship ***OCCUPIES*** between an agent and a position.	Class association named OCCUPIES.
		Class	**G'1. 5**	Relationship ***COVERS*** between a position and a role.	Class association named COVERS.

		Class	**G'1. 6**	Relationship ***PLAYS*** between an agent and a role.	Class association named PLAYS.
G2	Task	Method	G2.1	Tasks defined in SD model.	Methods with public visibility.
		Method	G2.2	Tasks defined in SR model.	Methods with private visibility.
G3	Resources	Class	**G'3. 1**	Resources defined in SD model.	Class if this dependence has the characteristics of an object.
		Attribute	**G'3. 1**	Resources defined in SD model.	Attribute with *private* visibility in class that represents the dependee actor if this dependence cannot be characterized as an object
		Attribute	**G'3. 2**	Resources (sub resources) defined in SR model.	Attribute with *private* visibility in the class that represents the actor in which the sub resource belongs (if this sub resource cannot be understood as an object).
		Class	**G'3. 2**	Resources (sub resources) defined in SR model.	An independent class, otherwise.
G4	(Soft)Goal	Attribute Boolean	G4.1	(Soft)Goals in SD model.	Attribute with public visibility in the class that represents the dependee.
		Attribute Boolean	G4.2	(Soft)Goals in SR model.	Attribute with visibility public in the class that represents the actor in wich the sub goal belongs.
G5	Relationship Task Decomposition	OCL	G5	Task Decomposition.	Represented by pre and posconditions (expressed in OCL) of the corresponding pUML operation.
G6	Relationship Means-End	OCL disjunctions of all possible means achieving the end.	G6.1	(Soft)Goals-(Soft)Goals.	the disjunction of the means values implies the *end* value.
		OCL	G6.2	(Soft)Goal – Task, Resource-Task.	The post-condition of the means task implies the value of end.
		OCL	G6.3	Task – Task.	The disjunction of the post-condition of the means imply the pos-conditions of the end.

Table 1 – Original Mapping Guidelines [13] x Extended Mapping Guidelines

In this paper we extend the previous guidelines [13] to cater for advanced structuring mechanisms. The previous guidelines and the proposed extensions are presented in the table 1. In particular we will extend guideline G1 (related to the mapping of i* actors to pUML classes) and the guideline G3 (related to the mapping of the *i** resources to pUML classes). The new guidelines are being denoted by **G'** symbol. From the i* models (Figure 1, 2 and 3) and with the enhanced guidelines we are able to construct the class diagram shown in Figure 4.

Guideline G'1.1: The i* actors (agents, roles or positions) can be mapped to pUML classes. OCL constraints can be attached to the actor-generated classes.

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

Guideline G'1.2: The i* relationship *IS-PART-OF* between actors can be mapped to a class aggregation in pUML.

Ex: 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.

Guideline G'1.3: The i* relationship *IS-A* between actors can be mapped to class generalization/specialization in pUML.

Ex: In our case study 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 to a class association in pUML named OCCUPIES.

Ex: 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.

Ex: 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, also an association between *Internet Sales* class and *CD Reservation* class is inserted, both associations 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.

Ex: 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*.

Guideline G3: *The i* resources can be mapped to pUML classes*.

Guideline G' 3.1: Resources defined in Strategic Dependency (SD) model can be mapped to pUML in two ways,

- A resource dependency can be mapped to a class in pUML if this dependence has the characteristics of an object as defined in the object-oriented paradigm. An association, with associations end that indicate who is the *depender*, is created between the class that represents the resource and the class that

represents the actor that depends on the resource. Another association, with association end that indicates who is the *dependee*, is created between the class that represents the resource and the class that represents the actor that is responsible for availability of the resource.

- A resource dependency can be mapped as an attribute with *private* visibility in the class that represents the dependee actor (agent, position or role) if this dependence cannot be characterized as an object as defined in the object oriented paradigm.

Ex: In Figure 3, the resource dependency *Personal Data* will be mapped as an attribute of the Client class (Figure 4) with visibility private.

Guideline G'3.2: Related to resources (sub-resources) defined in Strategic Rationale (SR) model.

- A sub resource defined in Strategic Rationale model can be mapped to an attribute with *private* visibility in the class that represents the actor (agent, position or role) in which the sub-resource belongs (provided this sub resource cannot be understood as an object). Otherwise, this resource will also be mapped as an independent class in pUML.

Ex: In Figure 3, the sub resource *Register Rules* belongs to the *Internet Sales* position will be mapped to an independent class in pUML (Figure 4).

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. 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. 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

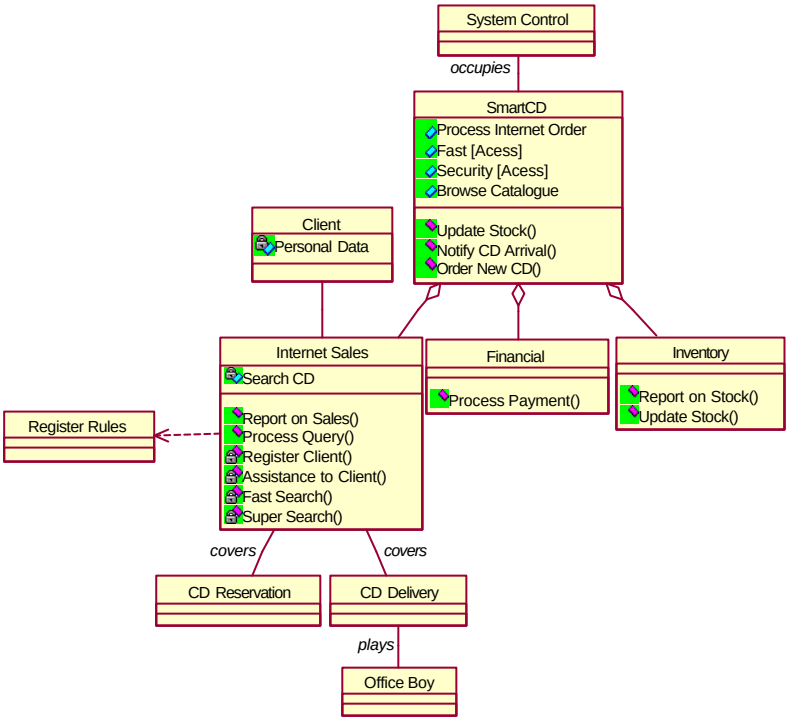

Fig. 4. Context Class Diagram of the SmartCD System

4. 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 describing the XGOOD tool.

4.1. Organizational Modeling Environment - OME

OME is a goal-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 Plugins. OME kernel has a layered architecture, comprised of three major modules (View Layer, Model Framework Layer and KB Layer). The KB (Knowledge Base) Layer is

responsible for the storage the objects used as a specific model, their relationships, and their attributes (pertinent to the model). The major module in the KB is a *Telos* [12] repository.

4.2. Extension mechanism for Rational Rose

The Rational Rose is a visual modeling tool that supports Object Oriented Modeling in UML. Rational Rose also 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 can 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.

4.3. e<u>X</u>tended GOOD (Goals into Object Oriented Development) Tool

GOOD is the prototype of a tool [16] that supports the automatic 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 GOOD tool [16] was written using a Rose Script Language, making it impossible to use it with UML modeling tools from different vendors. Moreover, certain concepts such as contribution links, refinement of actors, restrictions in OCL (Object Constrain Language) are not supported. Also, often certain elements of the organizational model are not part of the software system to be developed. Unfortunately, the GOOD tool does not offer ways to select which elements will (or not) be mapped. Hence, work is underway to extend the tool (XGOOD).

This new tool should be compatible with any modeling environment. This is possible through the adoption of technologies such as XMI (XML Metadata Interchange), which is an open standard, accepted and adopted since 1999 by the OMG (Object Management Group) [14], for the representation of models and metadatas. It integrates three standards: MOF (Meta Object Facility) [14], XML (eXtensible Markup Language) [17] and UML.

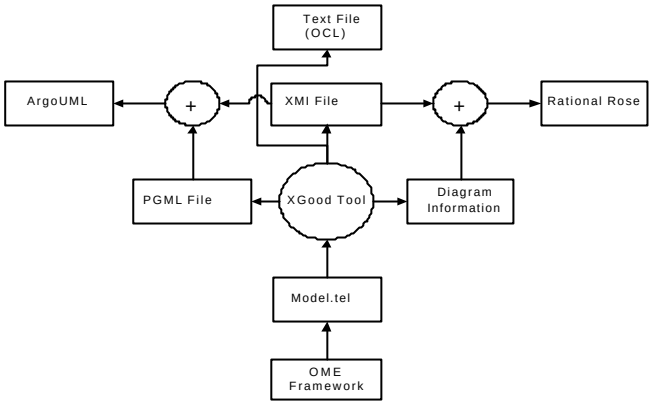

Fig. 5. The XGood Tool

The new XGOOD tool will map the i* model directly into the UML Class and/or Use Case Diagrams. The format used to represent the model will be XMI (see figure 5). Moreover, it will provide functionalities such as: selection of adequate guidelines, choice of elements to be included in the model as well as definition of new guidelines for the mapping. It will also be possible to include the information of the diagram inside of XMI file (for the case of the Rational Rose) or in a separate file (for the case of the ArgoUML). Support to OCL expressions will also be provided.

The XMI uses the syntax of the XML (eXtensible Markup Language) to represent the models in an file. The models are represented through tags and attributes. It also uses DTD (Document Type definition) and, more recently in version 2.0, XSD (XML Schema Definition) as a way to validate and to keep the consistency of the models. For example, in the Rational suite, the Unisys Rose XML Tools, is capable to import/export its models to XMI version 1.1. Similarly, the ArgoUML 0.14 [15] supports XMI version 1.0. Both of them use the DTD to carry through the validation of the data.

Unfortunately, the modeling tools generally possess a special and proprietor file format to save its models. This disables the sharing of the information. Hence, a tool developed for the Rational Rose, for example, would not function with the ArgoUML files. The XMI deals with only the information of the model, that is, it does not say anything regarding the information of the diagram. Each tool possesses its way for the representation of diagram. The Rational Rose includes one additional tag to XMI file: Diagramming.Diagram., which stores information such as type of the font and the position of the elements and their dimensions.

The ArgoUML [15] saves the information of the diagram in an PGML file (Precision Graphics Markup Language - with extension pgml) and the information of the model in an XMI file (with extension xmi) [15]. To import a XMI file produced in ArgoUML into the Rational Rose, we must add the information of the class diagram and/or the use case diagram to the XMI file (see figure 5).

The adoption of a standard, accepted and recognized by the OMG, brings a bigger flexibility to the XGOOD tool. The support of new modeling tools (for example the

Poseidon [18]) could be carried out without major effort, since these tools imports XMI files (see Figure 6). However, the biggest problem is the graphical representation of the models. Today, is still not possible, for example, to directly export a Rational Rose file to the ArgoUML.

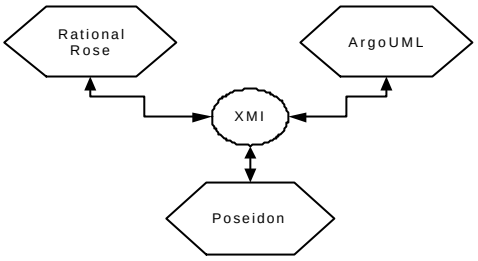

Fig. 6. The XMI interface

6. Related Work

The area of Requirements Engineering has developed several novel techniques for early requirements capture. 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 Actor Model, a Concept Model, and an Information System 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 [7] 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 the attributes related with quality, 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, and 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 late 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 real industrial case studies are also expected. Work is underway to extend the current tool support to cater for the new guidelines described in this paper (**G'1.1, G'1.2, G'1.3, G'1.4, G'1.5, G'1.6, G'3.1** and **G'3.2** guidelines).

References

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

[2]Booch, G., Jacobson, I. and Rumbaugh, J.: “Unified Modeling Language User Guide”. Rational Software Corporation. Addison-Wesley Object Technology Series. Jan., 1999.

[3]Mylopoulos, J., Chung, L and Yu, E.: `From Object-Oriented to Goal-Oriented Requirements Analysis' Communications of the ACM, 42(1): 31-37, January 1999.

[4] Warmer, Jos B. and Kleppe, Anneke G.: “The Object Constraint Language: Precise Modeling with UML”. Addison-Wesley Object Technology Series. March, 1999.

[5]Yu, E.: “Why Agent-Oriented Requirements Engineering”. Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, Pisa, Italy. E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1998. pp. 15-22.

[6] Precise UML Group, pUML:http/www.cs.york.uk/puml.

[7] van Lamsweerde, A.: “Requirements Engineering in the year 00: A Research Perspective”. Invited paper to ICSE’2000, in Proc. 22nd International Conference on Software Engineering, Limerick, June 2000.

[8] Boman, M., Bubenko, J., Johannesson, P. and Wangler, B. “Conceptual Modeling”. Prentice Hall Series in Computer Science. 1997.

[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. and 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. 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. F. B., Alencar, F. M. R., Cysneiro FILHO, G. A. A., Integrating Organizational Requirements and Object Oriented Modeling. In: Fifth International Symposium on Requirements Engineering - RE’01, Toronto. p.p. 146-153 2001.

[14] Object Management Group: “OMG XML Metadata Interchange (XMI) Specification”, 2002. http://www.omg.org/technology/documents/formal/xmi.htm.

[15] A. Ramirez, P. Vanpeperstraete, A. Rueckert, K. Odutola, J. Bennett, and L. Tolke: “ArgoUML User Manual A tutorial and reference description”, 2003. http://argouml.tigris.org/.

[16] Csneiros Filho, G.A A.: “Ferramenta para o Suporte do Mapeamento da Modelagem Organizacional em i* para UML”, (In Portuguese). Centro de Informática, Federal University of Pernambuco, Brazil, Master Thesis.,Ago, 2001.

[17] Extensible Markup Language (XML). http://www.w3.org/XML/

[18] Poseidon for UML. http://www.gentleware.com/ .