From Early Requirements Modeled by the i* Technique to Later Requirements Modeled in Precise UML

Fernanda Alencar[1], Jaelson Castro[2*], Gilberto Cysneiros[2], John Mylopoulos[3]

[1]Universidade Federal de Pernambuco, Dep. Eletrônica e Sistemas, Brazil
E-Mail: fmra@npd.ufpe.br

[2]Universidade Federal de Pernambuco, Centro de Informática, Recife, Brazil
E-Mail: {jbc,gaacf}@cin.ufpe.br

[3]University of Toronto, Department of Computer Science, University of Toronto, Canada
E-Mail: jm@cs.toronto.edu

Abstract. Requirements capture has been acknowledged as a critical phase of software development, 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. Unfortunately, the current dominant object oriented modeling technique, i.e. Unified Modeling Technique, is ill equipped for capturing early requirements which are typically informal and often focus on organisational objectives. UML is more suitable for later phases of requirements capture, which usually focus on completeness, consistency, and automated verification of functional requirements for the new system. In this paper, we present some guidelines for the integration of early and late requirements specifications. For the organizational modeling we use the *i** technique, which focuses on the description of organizational relationships among various organizational actors, as well as an understanding of the rationale for the alternatives chosen. For the functional requirements specification, we rely on the precise Unified Modeling Language (pUML), annotated with constraints described in OCL. A small CD store example is used to illustrate how the requirements process iterates between the early and late requirements specification.

Keywords: Requirements Engineering, Object Oriented Development, UML, Early and Later Requirements

* This work was carried out while the first author was visiting the Department of Computer Science, University of Toronto. The research was partially supported by the CNPq – Brazil under grant 203262/86-7.

1 Introduction

Often, software 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 software developers of the system, also the frequency of organizational changes which cannot be accommodated by existing software systems (or their maintainers). Hence, requirements capture has been acknowledged as a critical phase of software development, precisely because it is the phase which deals not only with technical knowledge, but also with organisational, 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 (Erikson and Penker, 2000). Consequently, there is a need for modeling and analysis of stakeholder interests and how they might be addressed, or compromised, by various system-and-environment alternative structures. Indeed, the Unified Method has suggested the following archetypal workflow for requirements capture (Jacobson et al.,1999):

- List candidate requirements,
- Understand system context,
- Capture functional requirements,
- Capture non-functional requirements

However, the production of high quality specifications is not easy. Usually the clients do not exactly know what they want and sometimes the requirements may not reflect the real needs of the clients. It is common for requirements to be incomplete and/or inconsistent.

Recent research on requirements engineering has drawn an important distinction between *early phase* requirements capture and *late phase* requirements capture (Yu, 1997). Early phase requirements activities are typically informal and address organisational or non-functional requirements. The emphasis is on understanding the motivation and rationale that underlie system requirements. Late phase requirements activities usually focus on completeness, consistency, and automated verification of requirements.

The Unified Modeling Language (Booch et al., 1999) 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 organisational 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 organisational objectives behind the software development project (Mylopoulos et al., 1999a). We argue that the *i** framework (Yu and Mylopoulos, 1994), is well suited for early-phase requirements cap-

ture, 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*. To model and understand issues of the application and business domain (the enterprise) a developer can use the *i** technique which allows a better description of the organisational relationships among the various agents of a system as well as an understanding of the rationale of the decisions taken. For the functional requirements specification, the developer can rely on UML, or if formality is required, the precise Unified Modeling Language (pUML) (Evans and Kent, 1999), annotated with constraints described in OCL (Warmer and Kleppe, 1999).

In this paper we present the transition from early (informal) descriptions in *i** to late (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 paper reports on work that aims at enriching the modeling power of UML. Section 2 introduces the language used for the early requirements description, namely *i** technique. In section 3, we provide some means for transforming *i** models into precise specifications in pUML/OCL. Late requirements specification is described in Section 4. Section 5 reviews some related work, while Section 6 concludes the paper with a summary of its contributions. Throughout the paper, a small CD store example is used to illustrate how the requirements process iterates between the early and late requirements specification.

2 Using *i** for Early Requirements Capture

In this section we will review the main concepts of the i* technique (Yu, 1997). i* offers a modeling framework that focuses on strategic actor relationships. 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* (Yu, 1998) caters to some of these more advanced concepts. It can be used for: (i) obtaining a better understanding of the organisational relationships among the various organisational agents; (ii) understanding the rationale of the decisions taken; and (iii) illustrating the various characteristics found in the early phases of requirements specification. The participants of the organisational setting are actors with intentional properties, such as, goals, beliefs, abilities and compromises. These actors depend upon each other in order to fulfil their objectives and have their tasks performed. The i* technique offers two models: The Strategic Dependency (SD) model, and the Strategic Rationale (SR) model.

2.1 The Strategic Dependency Model

The Strategic Dependency model 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 fulfil 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). We can also classify actors into *agents*, *roles* and *positions*. An agent is an actor with concrete physical manifestations (a person or a system). A role is an abstract characterization of the behaviour of a social actor within some specialized context, domain or endeavor. A position is a set of roles typically played by one agent. Finally, *i** supports the analysis of opportunities and vulnerabilities for different actors (Yu and Mylopoulos, 1994).

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 in 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.

At this early stage of requirements capture we have identified three actors: *Client*, *Store* and *SmartCD*. This last actor indeed 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 *Store* actor can be fond in Figure 1. The Client depends on the 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*. 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*). The Client expects the access to software system to be fast (softgoal *Fast[Access]*) and to use it to keep the stock updated (task *Update Stock*). Last but not least, the *Store* actor also has some expectations on the commissioned *SmartCD* system. It relies on the software system to process internet order (goal *Internet Order)* and to control its stock (task *Update Stock*).

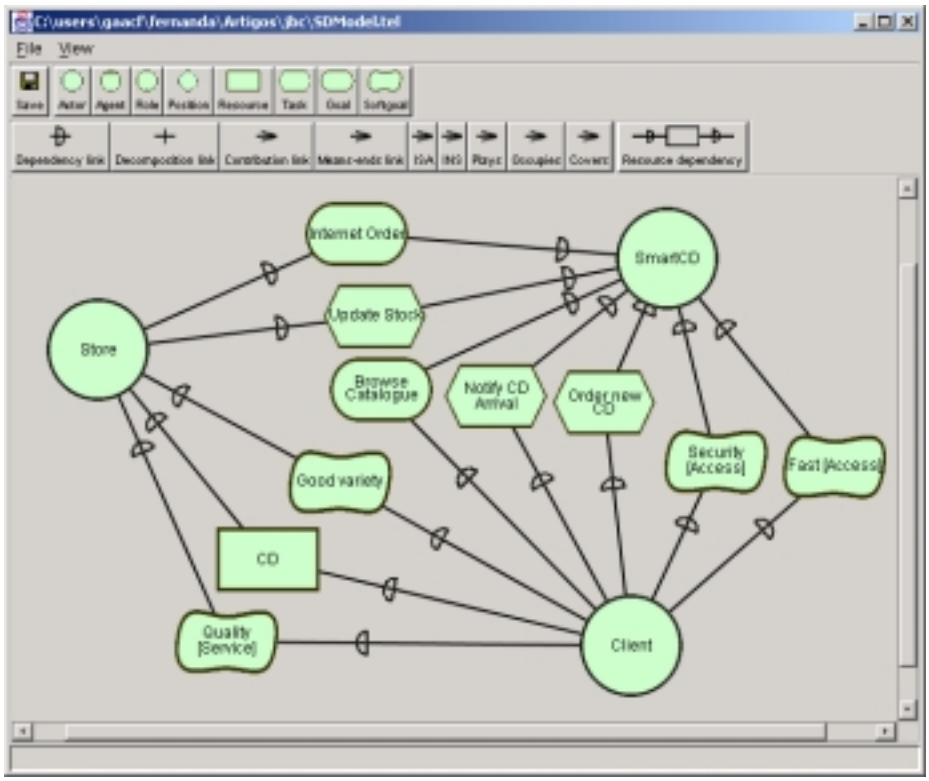

Fig. 1. Strategic Dependency Model

2.2 The Strategic Rational Model

The second model of the i* technique is the Strategic Rationale Model (SR). 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 2 we use SR notation to detail the *Store* and *SmartCD* agent. Due to space limitation we now only comment some aspects. An interested reader can find a fuller description of the approach in (Alencar, 1999).

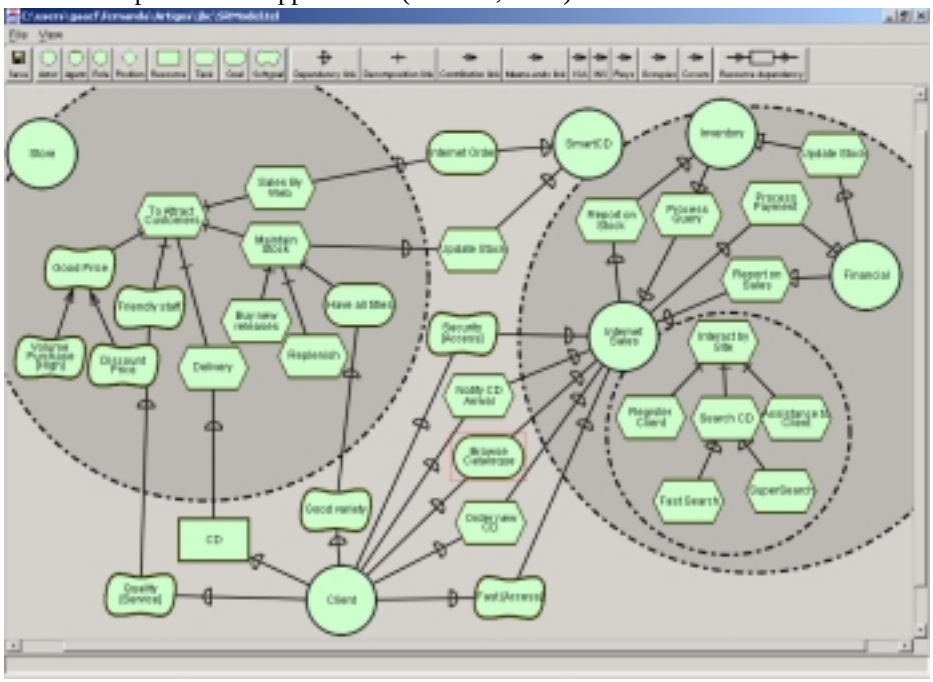

Fig. 2. Strategic Rationale Model

The store is interested in attracting (new and old) clients (expressed by task node *To Attract Customers*). Several strategic decisions were taken in consideration and as a result the task was decomposed into five aspects (expressed by a task-decomposition link):

- The need to offer reasonable prices (captured by softgoal Good Price). Two alternatives are considered for meeting this objective: to offer discount on selected items (softgoal Discount Price) or obtaining good deals by buying large quantities of popular CDs (softgoal Volume Purchase[High]),
- To need to establish a courteous relationship between the store staff and its clients (represented by softgoal Friendly staff). This is considered to be a way of meeting the quality of service expected by the client.
- To define standard procedures for delivering CD (expressed by sub-task Delivery),
- To be able to maintain a good stock of CDs (captured by sub-task Maintain Stock). This will also require the updating of the online information system.

Hence the task dependency Update Stock between the Store and SmartCD actors.

- To handle internet sales (captured by sub-task Sales By Web), which depends on the adequate software system, hence the goal dependency Internet Order between the Store and SmartCD actors.

After some considerations, it has been agreed that the task of maintaining the stock needs to be further decomposed:

- The overall objective is to have all available titles on stock (captured by goal *Have all titles*). If this is met, certainly the client will be very pleased (see the softgoal *Good Variety* between *Client* and *Store* actors),
- Buying new titles (expressed by a sub-task *Buy new releases*),
- Making sure that popular CDs are re-stocked (expressed by a sub-task '*Replenish*').

The SmartCD represents the information system that will help the store to accomplish some of its tasks strategic objective, namely to attract customers (see Figure 2). The system can be composed into various modules or sub-actors including: inventory, financial and the internet sales module (see Figure 2). These actors have some strategic dependencies each other.

The *Internet Sales* agent depends on the *Financial* agent to process the payment of the CDs bought by some Client for example, to verify the customer's bank information or to contact with the administrators of credit cards. This dependencies 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 *Client* agent will act basically on the *Internet Sales* agent. The *Financial* agent depends on the Internet Sales agent to inform the data of the sales, for example, Client's data, the purchase value, payment form, local of delivery, amount and type of product sold. This dependency is expressed by the task dependency "*Report on Sales*". The *Financial* agent 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* agent depends on the *Internet Sales* agent to request information on the Store's stock of products (the task dependency "*Process Query*").

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). Now, according to conventional software development techniques, the elements of the dependencies such a goal, a task or softgoal need to be operationalized before the end of late requirements analysis. We can then move to provide a detailed (functional) specification of system.

3 From Early to 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) (Evans and Kent, 1999) which provides a precise denotational semantics for core UML elements, such as: relationship, classifier, association, and generalization. The interested reader can visit the pUML site (Precise UML Group, 2000) for a complete description of the approach.

However, 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) (Warmer and Kleppe, 1999). OCL is a textual language, also part of the Object Management standard, that can precisely describe constraints for object oriented models.

Figure 3 shows the mapping of the CD Store model in i *, for a context class model based on the guidelines presented below. That model is just an intermediate model that serves as the basis for the class diagram of the system. It presents semantics classes derived from the problem domain (Monarchi, 1992).

In the sequel we suggest six heuristics for transforming i* based early requirements models to pUML/OCL-based late requirements:
Guideline G1: Related to actors;
Guideline G2: Related to tasks;
Guideline G3: Related to resources;
Guideline G4: Related to goals and soft-goals;
Guideline G5: Related to tasks decomposition;
Guideline G6: Related to means ends links.

Guideline G1:

Actors in the *i** framework, can be mapped to classes in pUML. OCL constraints can be attached to the actor-generated classes.

In our case study (see Figure 1) there were three actors: *Store*, *Client* and *SmartCD*. These actors can be mapped to the three classes shown in Figure 3.

Guideline G1.1:

Actor composition in *i** corresponds to class aggregation in pUML.

In our CD *Store* case study (see Figure 1), the Strategic Dependency contained three actors: *Store*, *Client* and *SmartCD*. In pUML (see Figure 3), CD Store class is the aggregate of three composite classes.

Guideline G2:

Tasks in *i**, are mapped to class operations in pUML.

Guideline G2.1:

A task dependency, between a depender and a dependee actor in the SD model, corresponds to a public operation in the dependee pUML class.

In our case study (see Figure 1), the *Store* Actor depends on the *SmartCD* Actor for updating its stock (task *Update Stock*). Similarly, the *Client* actor depends on the *SmartCD* Actor for two tasks: ordering CDs (*Order new CD*) and receiving notification of goods arrival (task *Notify CD arrivals*). Hence, in Figure 3 you can observe

that the *SmartCD* pUML class will be responsible for supporting the three (public) operations (Update Stock, Notify CD Arrival, Order New CD) .

Guideline G2.2:

A task in the SR model is mapped to a local operation in the corresponding pUML class.

In our case study, see Figure 2, a key task for the *Store actor* was to be able to *Attract Customers.* This consists of three subtasks: to handle internet orders (*Sales By Web*), to *Maintain Stock* and to deliver the CDs (*Delivery*). Maintaining stock included obtaining the new releases (*Buy new releases)* and renewing items (*Replenish*). Therefore the corresponding pUML class has five (local) corresponding operations (see Figure 3).

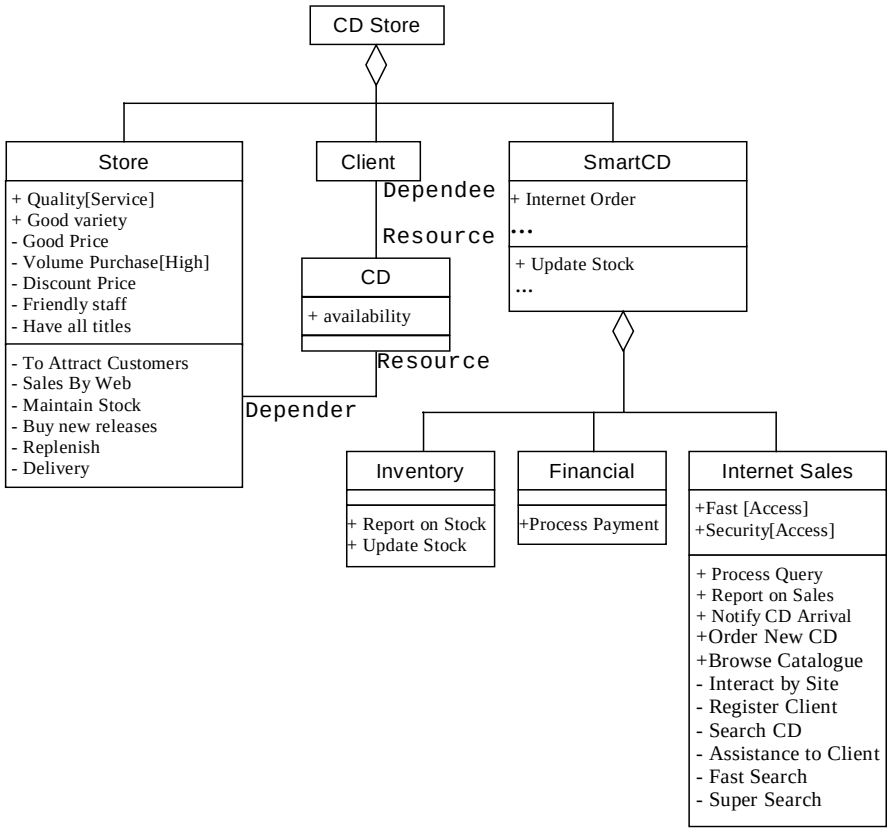

Fig. 3. Context Class Diagram of the CD Store

Guideline G3:

Resources in *i** are mapped as classes in pUML. A public attribute of the type Boolean, indicates the availability of resource.

In our example, see Figure 1, *Client* actor depends on the *Store* Actor, to obtain a CD resource. In Figure 3, we can observe that the *CD class* has been introduced to represent the resource. A boolean attribute (*availability*) indicates if is the resource is at hand.

Guideline G4:

Strategic goals and soft goals will be mapped to attributes of the type boolean and enumerated type, respectively, in pUML classes.

Goals are well defined, hence it is always possible to establish if one has been fulfilled or not. On the other hand, softgoals are not well defined. They can only be "satisfied" to some degree. Hence, an enumerated type is better suited for their representation in pUML/OCL, whose values represent different degrees of softgoal fulfillment.

Guideline G4.1:

Goals and soft goals dependencies in SD models are mapped to public boolean and enumerated attributes, respectively, of the dependee pUML class.

In our case study, the *Client* Actor expects that the *Store* Actor could have a good stock (*Good variety* softgoa*l)* and provide a good service (*Quality[Service*] softgoal). Therefore, in the corresponding pUML *Store* class, two enumerated attributes are added (see Figure 3).

Guideline G4.2:

Goals and soft goals dependencies in Strategic Rationale Models (SR), are mapped to local boolean and enumerated pUML class attributes, respectively.

For example, in Figure 2, we have that the *Store* actor has a well defined goal (to *Have all Title*), and four ill defined objectives or softgoals: to offer *Good Price*, to have *Volume Purchase* [High], to give *Discount Price*, and to have a *Friendly staff*. In Figure 3 we observe that these extra attributes have been included to *Store* Class.

Next guideline deals with task. Operations in pUML can be used to describe tasks performed by an actor. If we need to provide a more precise account of the operation, we can rely on OCL to specify its pre and post conditions. However, in the i* framework tasks can be decomposed into sub-task, sub-goal, sub-softgoal and sub-resource.

Guideline G5:

Task decomposition is represented by pre and post-conditions (expressed in OCL) of the corresponding pUML operation.

The pre-condition is the conjunction (AND OCL connector) of sub-tasks pre-conditions.

The post-condition is the conjunction (AND OCL connector) of all: (i) sub tasks post-conditions; (ii) *resource* Boolean attributes; (iii) *goal* Boolean attributes (iv) *soft-goal* enumerated attributes.

Store:: To Attract Customers
pre: pre-Delivery **and** pre-Maintain Stock **and** pre-Sales By Web
post: GoodPrice = 'value' **and** Friendly staff = 'value' **and**
post-Delivery **and** post-MaintainStock **and** post-Sales By Web

Fig. 4. Task decomposition in OCL

Consider for example, the task *To Attract Customers* (Figure 2). It is decomposed into three sub-tasks (Delivery, Maintain Stock and Sales By Web) and two sub-goals (*Good Price* and *Friendly staff*). Let us use the OCL assertions *pre-Subtask* and *post-Subtask* to indicate generic pre and post-conditions of a sub-task. Moreover, assume that the OCL assertion *value* indicates one of the possible values of the enumerated type (posit, negat, undef,) associated to a soft goal. Figure 4 shows the corresponding OCL description.

When we work with the later requirements we can refine the pre and post conditions of the three operations: Delivery, Maintain Stock and Sales by Web. This activity is typical of later phase of the development process. Hence, we can suggest some conditions for these operations as you can see in the Figure 5.

The SR models also provides for several types or means-end link. The "end" can be a goal, task, resource, or softgoal, whereas the "means" is usually a task (GT, TT, RT and ST links). Sometimes it is also useful to have means-end hierarchy of softgoals or goals (SS and GG links).

Store:: To Delivery
pre: delivery address and payment
post: delivered CD

Store:: To Maintain Stock
pre: quantity >= "value"
post: CD in stock

Store:: To Sales By Web
pre: site on line
post: increaseded sales volume

Fig. 5. Pre and Post Conditions Estimated

Guideline G6:
Means end-analysis is represented by OCL disjunctions of all possible means achieving the end.

Guideline G6.1(SS and GG Links):
If the *end* is a (soft) goal and the *means* are (soft) goals than the disjunction of the means values implies the *end* value.

In our case study (see Figure 2), there are two means of offering reasonable prices Negotiating discounts based on high volume purchase (softgoal *Volume Pur-*

chase[High]) or by promoting sales (softgoal *Discount Price*). Either way the end goal (softgoal *Good Price*) is achieved. In the Figure 6 the corresponding OCL representation is presented.

Store
Volume Purchase [High]='value' **or** Discount Price = 'value'
implies Good Price = 'value'

Fig. 6. Means-end analysis

Guideline G6.2 (GT, RT and ST Links):

If the *end* is either a goal, resource or softgoal and the *means* is a task than the *post-condition* of the *means* task *implies* the value of *end* goal (boolean) attribute, resource (boolean) attribute or softgoal (enumerated) attribute.

In our case study these means-end links did not occur.

Guideline G6.3 (TT Link):

If the *end* is a task and the means are tasks then the disjunction of the post-condition of the *means* task *imply* the post-conditions of the *end* task.

In our case study this means-end link did not occur.

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 organisational model are not part of the software model, since not all of the organisational 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 organisational model. Nonetheless, as we shall see, pUML/OCL also can be used to represent this information.

So far we have been able to identify a context class diagram for problem at hand. Now we can proceed to give some more details of the software system to be developed.

4 Late Requirements

As shown in subsection 2.2, the SmartCD actor represents the information system that will help the store to accomplish some of its tasks strategic objective, namely to attract customers (see Figure 2). The system was decomposed into three modules or sub-actors including: inventory, financial and the internet sales module.

Due to space limitation we will concentrate on *Internet Sales Module*, since it is related to the *Sales By Web* task, shown in the strategic rationale model of Figure 2. It is part of the system responsible for the process of sales through the Internet. We will consider the context class diagram of Figure 3 as the starting point of our discussion.

The SmartCD represented by the *Internet Sales Module* will interact with the clients through a site. This site allows the visitors to search for CDs, see information about pop stars and news about the musical events.

There are two ways for a client to search for a CD: the *fast* and *super* search. In the *fast* mode the visitor informs the name of the album, of the artist, or of the music. In the super search the objective is to help those that still do not know which title to buy. For that *super* search type the following options are available: name of the album or of the music, the styles (Pop, Rock, Rap, Reggae, Jazz, etc.), the recording, the repertoire (national or international) and the release time.

Assistance to clients can be provided upon request by e-mail or through a FAQ page. The FAQs (Frequentely Asked and Questions) contains answers for the most common questions. If the visitor does not find the appropriate answer he/she can fill in form (including the subject, name, number of the purchase request) and submit it. Upon receipt, an on-line sellers will answer the question.

A visitor (if not already a client) would have to register in order to use the system. The register operation is available through a page where the client supplies its personal data (complete name, identification no. , gender, birthday), an access name (login), a password, a note (to help to recollect the password), its complete address (street, number, neighborhood, city, state and zip code), residential and commercial telephone, e-mail, the address for delivery (street, number, neighborhood, city, state, zip code, telephone for contact) and the payment form. In particular we are conceiving two forms of payment: credit card or direct debit in checking account. For the payment by credit card it is necessary the title-holder's name, the type of the credit card, the number of the card and validity of the card. For the direct debit in checking account it is necessary the title-holder's name, the name of the bank, the number of the agency and the number of the checking account. If necessary the client can later revise his/her personal information. For reasons of security the client can choose to fax or e-mail the personal information.

The site offers to its clients special services, for example: security in the transactions and personalized attendance. For the sake of total security the customers' data are stored in a safe and isolated server (not connected to the Internet), with restricted access to authorized employees. SmartCD will use a safe communication protocol (SSL - Secure Sockets Layer). Client profile can also be provided. It consists of client's preferences such as musical styles (Pop/Rock, Blues/Jazz, Infantile, Samba/Pop,...), artists or favorite groups and the desire or not of receiving information on promotions, releases, etc.

Based on these late requirements, a revised UML class diagram can be obtained (see Figure 7). Some classes, attributes and methods represented in the fig. 3 can be used in a direct way while others can be refined. The model of the fig. 7 display the part of the class model of the SmartCD module. Note that classes CD, Inventory and Client present in fig. 3 are also represented in fig.7. The method Update Stock of the class Inventory in fig.3 corresponds to the methods removeCD and insertCD of the class Inventory in fig.7.

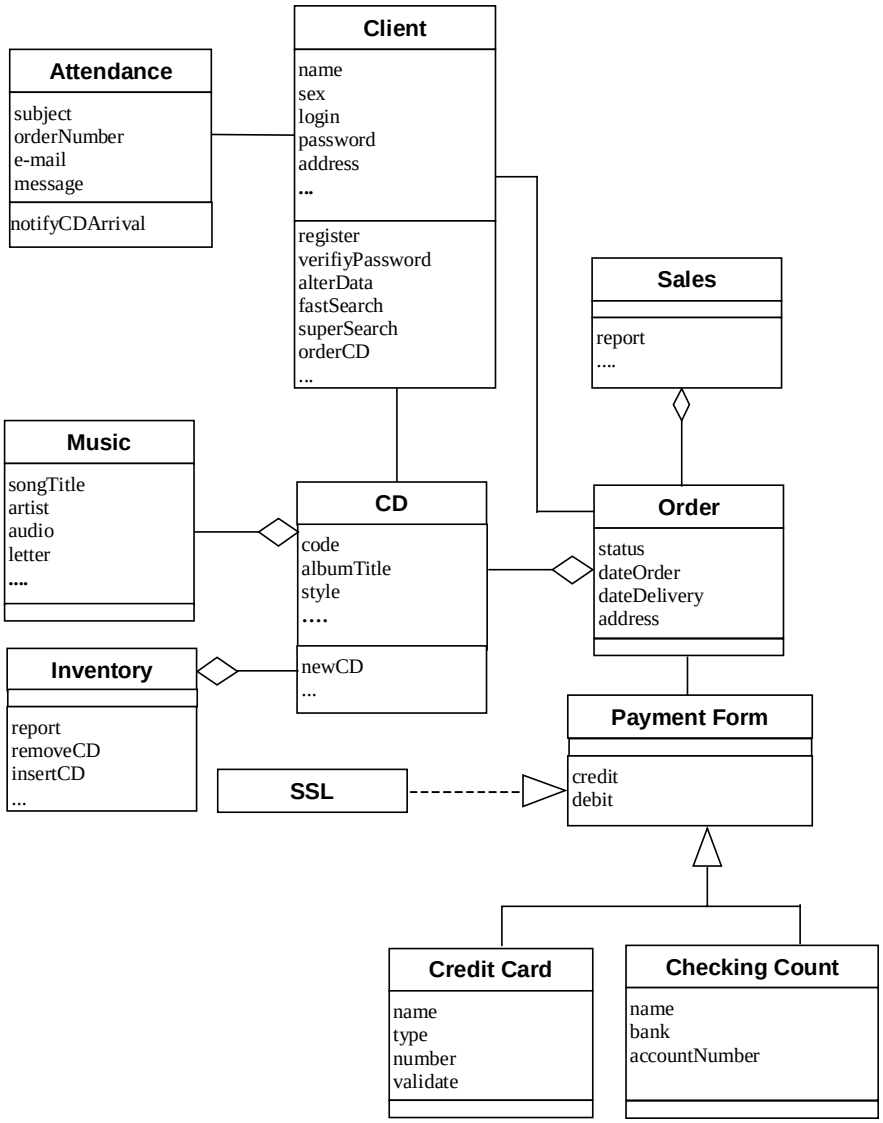

Fig. 7. Class Diagram of the SmartCD with WEB Module

5 Related Work

The area of Requirements Engineering (van Lamsweerde, 2000) has developed several novel techniques for early requirements capture (Boman et al., 1997),(van Lamsweerde et al., 1998). Bubenko emphasizes the need to model organizations and

their actors, motivations and reasons (Boman et al., 1997). 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.

Another related work is the requirements modeling framework for manufacturing systems (MS) presented in (Petit,1999). It relies on two major ideas: a multi-formalism approach, combining several languages into a coherent formalism, and a component-based modeling approach. The modeling framework proposed combines the Albert II, i* and CIMOSA languages. The combination is achieved through meta-modeling and the definition of a set of mapping rules that establish a correspondence among some of the concepts of the three formalisms.

In the KAOS framework (van Lamsweerde et al., 1998) 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. Goals are reduced through means-ends reasoning to arrive at responsibilities for agents. The modeling of agents is specificational and prescriptive. Since agents are assumed to conform to prescribed behavior, one cannot easily analyze dependencies for opportunities and vulnerabilities. On the other hand, i* models offer a number of levels of analysis, in terms of ability, workability, viability and believability. These are detailed in (Yu97).

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 (Chung et al., 2000) 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 organisational and technical objectives (Mylopoulos et al., 1999b).

Although UML has been used mainly for modeling software, recent proposals have used it for describing enterprise and business modeling. For example, (Eriksson and Penker, 2000) 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 behaviour. In (Marshal, 2000) 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.

6 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 organisational 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 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.

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 organisational model are not part of the software model, since not all of the organisational 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 organisational model. Nonetheless, pUML/OCL also can be used to represent this information.

Further research is still required to handle some structuring concepts found in the i* framework, such as agent, role and position. To improve the integration of organizational and functional requirements Use Cases diagrams can also be considered (see Santander and Castro, 2000). Some real industrial case studies are also expected. Work is underway to provide some tool support for the mapping. The tool will import organizational requirements specification produced by the OME toolset and generate the corresponding pUML business model.

References

Alencar, F.: Mapping Organizational Modelling into Precise Specification (In Portuguese). Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil, Ph. D. Thesis (1999)

Boman, M., Bubenko, J., Johannesson, P. Wangler, B.: Conceptual Modeling. Prentice Hall Series in Computer Science (1997)

Booch, G., Jacobson, I. Rumbaugh, J.: Unified Modeling Language User Guide. Addison-Welsley Object Technology Series (1999)

Chung, L., Nixon, B., Yu, E., Mylopoulus, J.: Non-Functional Requirements in Software Engineering. Kluwer Publishing (2000)

Erikson, H., Penker, M.: Business Modeling with UML: Business Patterns at Work. OMG Press John Wiley & Sons (2000)

Evans, A., Kent, S.: Core Meta-Modelling Semantics of UML: The pUML Approach. UML'99 – The Unified Modeling Language: Beyond the Standard – The Second International Conference. Eds. Robert France and Bernhard Rumpe. Fort Collins, CO, USA (1999) 140-150

Jacobson, I., Booch, G. Rumbaugh, J.: Unified Software Development Process. Addison Weley Object Technology Series (1999)

Marshal, C.: Enterprise Modeling with UML: Designing Sucessful Software through Business Analysis. Addison-Wesley Object Technology Series (2000)

Mylopoulos, J., Chung, L., Yu, E.: From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM, 42(1): (1999a) 31-37

Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Extending Object-Oriented Analysis to Explore Alternatives. Submitted for publication (1999b)

Monarchi, D. et. al.: A Research Typology for Object-Oriented Analysis and Design. Comunications of the ACM, 35 (1992) 35-47

Petit, M.: Formal Requirements Engineering of Manufacturing Systems: A Multi-Formalism and Component-Based Approach. Computer Science Departament University of Namur, Namur, Belgium, Ph. D. Thesis (1999)

Precise UML Group. PUML get by Internet url: http/www.cs.york.ac.uk/puml (2000)

Santander, V., Castro, J.: Desenvolvendo Use Cases a partir de Modelagem Organizacional (In Portuguese). To Appear in III Workshop of Requirement Engineering (2000)

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 (1998)

van Lamsweerde, A.: Requirements Engineering in the year 00: A Research Perspective. Invited paper to ICSE'2000. To appear in Proc. 22nd International Conference on Software Engineering, Limerick, (2000)

Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley Object Technology Series (1999)

Yu, E., Mylopoulos, J.: Understanding 'Why' in Software Process Modeling, Analysis and Design. Proceedings Sixteenth International Conference on Software Engineering, Sorrento, Italy (1994)

Yu, E.: Towwards Modelling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of IEEE International Symposium on Requirements Engineering – RE97 (1997)

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) 15-22