Conceptual Schemas Generation from Organizational Models in an Automatic Software Production Process

By Alicia Martinez Rebollar

PhD Thesis

Presented to the Department of Information Systems and Computation of the Valencia University of Technology, Spain, and to the Department of Information and Communication Technology of the University of Trento, Italy in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

September 2008

The research reported in this thesis has been financially supported by the SUPERA project in Mexico, the Valencia University of Technology, Spain, the enterprise CARE Technologies S.A., and the University of Trento, Italy.

Printed in Spain
Valencia

Thesis Advisors:
Dr. Oscar Pastor Lopez, Valencia University of Technology, Spain
Dr. Paolo Giorgini, University of Trento, Italy

Thesis Committee:
Dr. John Mylopoulos, University of Trento, Italy
Dr. Xavier Franch, University of Catalonia, Spain
Dr. Jaelson Castro, University of Pernambuco, Brazil
Dr. Juan Sanchez, Valencia University of Technology, Spain
Dr. Vicente Pelechano, Valencia University of Technology, Spain

Abstract

At the present time, software engineering has proposed many techniques to improve the software development process, but the final goal has still not been satisfied. In most cases, the final software product does not satisfy the real needs of the final customers of the business where the system will be operated.

One of the main issues of current research works is the lack of a systematic approach to map each modeling concept of the problem domain (organizational models), into the corresponding conceptual elements of the solution space (object-oriented conceptual models).

The main objective of this thesis is to provide a methodological approach that enables the generation of conceptual and requirements models from organizational descriptions. We use three different, but complementary disciplines (organizational modeling, software requirements and conceptual modeling) in order to achieve this objective.

The thesis describes a requirements elicitation process that enables analysts to create a business model that represents the current situation of the enterprise. We consider that this model, which reflects how the enterprise currently implements its business processes, is the correct source to determine the expected functionality of the system-to-be. A process to identify the elements that are relevant to be automated from the business model is also proposed in this work. As a result of this process, an intermediate model is generated in order to represents the software system requirements.

Finally, we present a set of systematic guidelines to generate an object-oriented conceptual schema from the intermediate model. We also explore the generation of a late requirements specification from the intermediate model as an alternative solution for the thesis objectives. A specific object-oriented conceptual modeling case tool (OO-Method) is used to detail the software requirements of the system-to-be. The OO-Method case tool has also been used to deal with the aspects that are associated to the generation of object-oriented conceptual schemas.

The main contribution of the thesis is to make the model transformation process systematic by proposing a model-driven based approach that uses rules, algorithms and patterns to derive both an object-oriented conceptual model and a requirements model from the organizational context.

Resumen

Actualmente, la ingeniería de software ha propuesto múltiples técnicas para mejorar el desarrollo de software, sin embargo, la meta final no ha sido satisfecha. En muchos casos, el producto software no satisface las necesidades reales de los clientes finales del negocio donde el sistema operará.

Uno de los problemas principales de los trabajos actuales es la carencia de un enfoque sistemático para mapear cada concepto de modelado del dominio del problema (modelos organizacionales), en sus correspondientes elementos conceptuales en el espacio de la solución (modelos conceptuales orientados a objetos).

El principal objetivo de esta tesis es proveer un enfoque metodológico que permita generar modelos conceptuales y modelos de requisitos a partir de descripciones organizacionales. Se propone el uso de tres disciplinas, distintas pero complementarias (modelado organizacional, requisitos de software y modelado conceptual) para lograr este objetivo.

La tesis describe un proceso de elicitación de requisitos que permite al usuario crear un modelo de negocios que representa la situación actual del negocio (requisitos tempranos). Nosotros consideramos que este modelo, el cual refleja la forma en la que se implementan actualmente los procesos de negocio, es la fuente correcta para determinar la funcionalidad esperada del sistema a desarrollar. Se propone también un proceso para identificar los elementos que son relevantes para ser automatizados a partir del modelo de negocio. Como resultado de este proceso se genera un modelo intermedio que representa los requisitos del sistema de software.

Finalmente, presentamos un conjunto de guías sistemáticas para generar un esquema conceptual orientado a objetos a partir del modelo intermedio. Nosotros también exploramos, como solución alternativa, la generación de una especificación de requisitos tardíos a partir del modelo intermedio.

En esta tesis, una herramienta CASE para modelado conceptual orientado a objetos (OO-Method) ha sido utilizada para detallar los requisitos del sistema a desarrollar. Esta herramienta ha sido también

utilizada para tratar los aspectos relativos a la generación de esquemas conceptuales orientados a objetos.
La principal contribución de la tesis es hacer el proceso de transformación sistemático proponiendo un enfoque basado en modelos, el cual usa reglas, algoritmos y patrones para derivar el modelo conceptual y de requisitos a partir del modelo organizacional.

Sommario

Nella attualità, l' ingegneria del software ha proposto molte tecniche per migliorare il processo di sviluppo di software, ma l'obiettivo finale non è ancora stato soddisfatto. Nella maggior parte dei casi, il prodotto di software definitivo non soddisfa le reali esigenze dei clienti finali delle imprese in cui il sistema sarà gestito.

Uno dei principali problemi degli attuali lavori di ricerca è la mancanza di un approccio sistematico per mappare ogni concetto di modellazione al problema di dominio (modelli organizzativi), nei elementi concettuale corrispondenti dellao spazio di soluzione (modelli concettuali object-oriented).

L'obiettivo principale di questa tesi è di fornire un approccio metodologico che consente la generazione di requisiti concettuali e modelli organizzativi da descrizioni. Usiamo tre discipline diverse, ma complementari (modellazione organizzativa, requisiti software e modellazione concettuale), al fine di raggiungere questo obiettivo.

Questa tesi descrive un processo di elicitazione di requisiti che consente al utente di creare un modello di negozio che rappresenta la situazione attuale. Riteniamo che questo modello, che riflette su come l'organizazzione attualmente implementa i suoi processi di negozio, è la sorgente corretta per determinare la funzionalità richiesta del sistema. Si propone un processo per identificare gli elementi che sono pertinenti per essere automatizzati da il modello di negozio. Come risultato di questo processo, un modello intermedio è generato che rappresenta i requisiti del sistema di software.

Infine, vi presentiamo una serie di linee guida sistematiche per generare un schema concettuale object-oriented dal modello intermedio. Abbiamo anche esplorato la generazione di una specifiche di esigenze tardive del modello intermedio come una soluzione alternativa per gli obiettivi di questa tesi .

Uno strumento CASE per la modellazione concettuale orientata ad oggetti viene usata per dettagliare i requisiti del sistema a sviluppare. Questo strumento è stato utilizzato anche per affrontare gli aspetti che sono associati alla generazione di schemi concettuali orientati ad oggetti.

Il principale contributo della tesi è quello di rendere il modello di processo di trasformazione sistematica, proponendo un approccio basato sui modelli, che utilizza regole, modelli e algoritmi per derivare sia un modello concettuale e di un modello di requisiti dal contesto organizzativo.

Resum

Actualment, la enginyeria del programari ha proposat múltiples tècniques per millorar el desenvolupament de programari, no obstant això, la meta final no ha estat satisfeta. En molts casos, el producte programari no satisfà les necessitats reals del clients finals del negoci on el sistema ha d'operar.

Un dels problemes fonamentals dels treballs actuals és la manca d'un enfocament sistemàtic per fer correspondre cada concepte de modelització del domini del problema (models organitzacionals), en els elements conceptuals en l'espai de la solució (models conceptuals orientats a objectes).

L'objectiu principal d'aquesta tesi es promoure un enfocament metodològic que permeta generar models conceptuals i models de requisits a partir de descripcions organitzacionals. Es proposa l'ús de tres disciplines, distintes però complementaries (modelat organitzacional, requisits de programari i modelització conceptual), per assolir aquest objectiu.

Aquesta tesi descriu un procés d'elicitació de requisits que permet a l'usuari crear un model de negocis que representa la situació actual del negoci (requisits primerencs). Nosaltres creiem que aquest model, que reflecteix la forma en la que se implementen avui els processos de negoci, és la font adequada per determinar la funcionalitat esperada del sistema a desenvolupar. A més, es proposa un procés per identificar els elements que són rellevants per a ser automatitzats a partir del model de negoci. Com resultat d'aquest procés es genera un model intermedi que representa els requisits del sistema de programari.

Per últim, presentem un conjunt de guies sistemàtiques per a generar un esquema conceptual orientat a objectes a partir del model intermedi. Com a solució alternativa també explorem la generació d'una especificació de requisits tardans a partir del model intermedi.

Per a detallar els requisits del sistema a desenvolupar la tesi empra una eina CASE per la modelització conceptual orientada a objectes (OO-Method). Aquesta eina ha estat també utilitzada per a tractar els

aspectes relatius a la generació d'esquemes conceptuals orientats a objectes.
La principal contribució de la tesi és fer el procés de transformació sistemàtic proposant un enfocament basat en models, el qual empra regles, algoritmes i patrons per derivar el model conceptual i de requisits a partir del model organitzacional.

Acknowledgments

In January 2001, Hugo and I began this adventure. We never imagined all the hard and beautiful moments that we were about to live. Really I am very lucky, because I have known many great and wonderful people. Today they are my friends. Besides, I have had the opportunity to work in well-known research groups. This has given an addition value to this research work.
I would like to thank many people, because this thesis could not have been completed without their valuable support. First of all, I would like to thank my advisor Oscar Pastor, for his great interest in, and support of, the work that led to this thesis. He has taught me, primarily by his own example, how to conduct high-quality research at an international level. This work would not have been possible without his human, technical and financial support, but mainly I would like thank him for his friendship along these years. I would also like to thank my advisors John Mylopoulos and Paolo Giorgini for providing a great feedback on my work during my research stay in the University of Trento. A special thank is also extended to Jaelson Castro for his contributions, advice and friendship.
My gratitude goes also to all committee members: John Mylopoulos, Jaelson Castro, Xavi Franch, Juan Diaz, Vicente Pelechano as well as to all my colleagues of the OO-Method Research Group: Nelly, Isabel, Manoli, Joan, Jorge, Pedro, Marta, Javier, Gonzalo and Victoria.
I must thank the one person without whom none of this would have happened: my husband and colleague Hugo Estrada, He was who motivated me to begin this adventure. He is my everlasting source of strength, encouragement, love and happiness. Thank you, because you are part of my life. You have given me Denisse and Iker. I must also to thank them, because they have given me: strength, patience, love and fondness. Besides, they help me to think that life cannot be only work. Thank you my loves. You are all that I need to be happy.

Last but not least, I have to thank my mother *Teresa* for her support during these years of study. Also I would like to thank my brothers who let me monopolize our mother in the last years.

Thank you all.
Alicia Martinez Rebollar

Conceptual Schema Generation from Organizational Models in an Automatic Software Production Process

Contents

Chapter 1

Introduction

The main objective of this thesis is to provide a methodological approach that enables the generation of conceptual and requirements models from organizational descriptions. We use three different, but complementary disciplines (organizational modeling, software requirements and conceptual modeling) in order to achieve this objective.
The thesis describes a requirements elicitation process that enables analysts to create a business model that represents the current situation of the enterprise. We consider that this model, which reflects how the enterprise currently implements its business processes, is the correct source to determine the expected functionality of the system-to-be. A process to identify the elements that are relevant to be automated from the business model is also

proposed in this work. As a result of this process, an intermediate model is generated that represents the software system requirements. Finally, we present a set of systematic guidelines to generate an object-oriented conceptual schema from the intermediate model. We also explore the generation of a late requirements specification from the intermediate model as an alternative solution for the thesis objectives. A specific object-oriented conceptual modeling case tool (OO-Method [Past01]) is used to detail the software requirements of the system-to-be. The OO-Method case tool has also been used to deal with the aspects that are associated to the generation of object-oriented conceptual schemas.

The main aim of the thesis is to make the model transformation process systematic by proposing a model-driven based approach that uses rules, algorithms and patterns to derive both an object-oriented conceptual model and a requirements model from the organizational context.

Section 1 of this Chapter discusses the purpose of this research work. Section 2 presents the problem statement that we try to solve and the proposed solutions. Section 3 presents the research goals. Section 4 presents the context in which the thesis was developed. Section 5 presents the research design. Finally, section 6 outlines the structure of the thesis.

1.1 Motivation

Building information systems is currently a very difficult task [Thay02]. Many of the research studies in software engineering have been done to ensure the correct construction of software products. In this sense, Software Engineering provides a wide range of techniques that aim at improving the quality in the software development process: i.e., software requirements analysis, software design, novel programming methods, verification and validation tests, software configuration management, software quality insurance, analysis and design methods, planning, projects scheduling, programming languages, etc [Garz02]. All the techniques, methodologies and tools of this kind have been proposed in order to develop correct and usable software systems [Pres03].

Software Engineering has proposed many techniques to improve the software development process, but the final goal has still not been satisfied. In most cases, the final software product does not satisfy the real needs of the final customers of the business where the system will be operated. A good example of this is the great investment made in the CASE technology in the late 1980s and early 1990s. Many organizations that invested in CASE tools found that they had no significant effect on the productivity or quality of their products [Koto98].

The CASE tools changed the process for building software systems, increasing productivity by reducing the time associated to the software implementation. However, the current CASE tools do not address the real problems that these organizations were facing, such as the requirements engineering problems [Koto98]. Kotonya attributes some of these problems to [Koto98]:

- **Lack of stakeholder involvement:** the process does not identify or take into account the real needs of the stakeholders that are involved in the system. This problem can be addressed by including explicit activities concerning stakeholder identification.
- **Business needs are not considered:** The requirements engineering process is seen as a technical procedure rather than as a business-based process. This can lead to software requirements that do not satisfy the real needs of the business.
- **Lack of requirements management:** The process does not include effective techniques for requirements management. This means that changes to the requirements may be introduced *ad hoc* and that a great deal of time and effort may be required to understand and incorporate these requirements changes.
- **Lack of defined responsibilities:** The different people involved in the requirements engineering process may not fully understand their responsibilities. This means that some

tasks may not be carried out at all because everyone assumes that someone else is responsible for it.

- **Stakeholder communication problems:** The different stakeholders in the system (end-users, managers, engineers, etc.) fail to communicate effectively so that the resulting requirements document is not understandable (and hence verifiable) by all the stakeholders. This results leads to the implementation of incorrect or incomplete requirements, which may only be discovered after the system has been implemented.

However, this does not mean that current methodological proposals do not provide appropriate solutions for developing a software system, because they have been designed keeping in mind the specification of the technical properties of the software-to-be. We consider that the aspect that has been most neglected in the current CASE tools is that these techniques do not take into account the sources of the software system functionalities, which is directly correlated with business objectives and processes.

In this context, we agree with Bubenko, Jacobson and Rational [Bube94], [Jaco95a], [Rati02] on the importance of understanding the organization before beginning the construction of a software system. Emphasis must be placed on the following as a basis for building the software system: the identification of the environment in which the software system will work; the roles and responsibilities of the employees using the system; and the “things” that are handled by the business.

These authors [Rati02] [Bube94] [Giog05] [Jaco95a] argue that some of the key questions that need to be considered for the success of a software system are the following: where the system-to-be will be used, whom it will be used by, how it needs to be integrated with existing systems, which tasks it will automate, and under which circumstances it will be executed.

These kinds of questions can only be answered by conducting an analysis of the organizational setting. This will allow us to produce an information system that adds real value to the enterprise where the system will operate.

Within the scope of works that explore the use of organizational models in software engineering, we can find business engineering proposals [Jaco95a], which is a set of techniques to design business processes according to the specification of the goals of the enterprise. The business engineering techniques include:

- Procedures for design of the business.
- Notations that describe the design.
- Heuristics or pragmatic solutions to find the correct design, which is measured in terms of the specification of goals.

All mechanisms of this kind enable software analysts to better elicit the requirements of the system-to-be by showing which aspects should be automated. Therefore, the requirements that were elicited will manage the development of information systems that are correctly adapted to the organizational setting and that offer the appropriate functionalities to the final users. Although consensus exists about the relevance of using organizational knowledge as the correct source for determining software requirements, at the present time, only a few research efforts are focused on the problem of systematically reducing the real impedance mismatch between the software system and its operational environment. This non-correspondence makes it impossible for the information system to have the necessary functionality to permit the organizational actors to perform their organizational tasks. Thus, we consider that the problem of methodologically joining the business engineering area with software engineering has not yet been solved.

1.2 Problem statement

In the software engineering context there are interesting proposals such as [Past01], [Insf03], [Cock01], [Kula03], and [Oliv03] that methodologically guide the translation of the problem space (represented as high abstraction models that represent the static and dynamic system structure) to the solution space (represented as software representations).

On the other hand, in the business engineering context, only a few research works have been proposed to solve the problem of obtaining software specifications from organizational models [Bider02], [Cast02], [Fuxm03], [Koub00], [Kolp03], [Alen03], [Sant02], and [Orti01]. These proposals are focused on specifying the basic primitives that should be taken into account when a business model is specified. Some of the issues that are addressed by these proposals are: how to determine the primitives of the business patterns, how to represent them, and how to be able to insert this business-based modeling process into a traditional software production process. In this context, some authors [Yu97] [Louc98] [Cast02] distinguish between the early requirements phase (business engineering) and the late requirements phase (software engineering).
The main issue of current research works in this area is the lack of a systematic approach to map each modeling concept of the problem domain (organizational models) into the corresponding conceptual elements of the solution space (object-oriented conceptual models).
The goal of our proposal is to derive the late requirements phase from the early requirements phase in order to correctly map the organizational actor needs with the functionalities of the information system. Thus, software engineering will solve the problems associated with improving the quality of the generated software, while business engineering will solve the problems associated with understanding the environment in which this system will operate, understanding the roles and responsibilities of the employees who will use the system, and the "things" that are handled by the business [Jaco95a].
Our premise is that the solution to systematically joining organizational modeling with software specifications must include the following characteristics:

- The method must provide a clear understanding of the organizational environment where the system will operate. It must both identify what the users do before using the software system as well as understand *how*, by *whom*, and under which circumstances the system will be used in the organization setting.

- The method must provide the analyst with techniques to perform the software development process in a systematic and precise way, putting emphasis on the specification of the software system. The method must also provide the analyst with complete code generation mechanisms.

The main contribution of this thesis is to improve the software development process by providing a deeper understanding of the activities and goals of the business. Two well-founded approaches have been combined to fulfill this objective:

- The OO-Method approach [Past01], which is an automatic production process that automatically generates *complete* object-oriented systems based on the information contained in the conceptual models. The OO-Method is used to deal with the specification of requirements and conceptual models.
- The Tropos Framework [Bres04], which is a software development methodology that is based on intentional concepts, such as those of actor, goal, (goal, plan, resource, *softgoal*) dependency, etc. It uses these concepts as a foundation to model early/late requirements, architectural design, and detailed design.

Although the method presented in this thesis has been applied in the context of a specific software production process (OO-Method), the solutions could be extensible to other requirements modeling environments or conceptual modeling environments.

1.2.1 Requirements model

The main goal of requirements modeling techniques is to define the functionality of a software system [Kula00]. One of the most popular techniques for requirements engineering is use case modeling, which describes the functionality of an information system from the point of view of the system users [Cock01], [Sanc03], [Cons99]. Other proposals, [Insf02a], [Robe99], deal with requirements modeling using other design techniques, such as sequence diagrams, state transition diagrams, or requirements specification templates. The main idea in requirements modeling is to obtain complete processes

of requirements in order to obtain the expected functionality of the information system-to-be.
The main drawback of the current requirements techniques is that they only respond "*what*" actions the software system must execute. However, these techniques cannot give an answer to "*why*" the software system must be built.
McDermind [McDe94] indicates that when the functional specification of the software system is the focal point of the requirements analysis, requirements engineers tend to establish the scope of the software system before having a clear understanding of the user's real needs. This is why many of the systems developed from a requirements model that focuses only on the functionality of the software system do not comply with their correct role within the organization. Therefore, in a software production process that does not have the organizational processes modeling as the first stage, any effort to generate a prototype of an information system will not be able to assure the utility of the software system in the context of the organizational tasks.
Therefore, one of the purposes of this proposal is to use organizational models as the starting point to obtain a requirements model of the information system. To do this, we propose systematic guidelines to help analysts to detect the relevant organizational plans to be automated, and to use this information to generate a use-cases-based requirements model.

1.2.2 Conceptual modeling

The traditional way of engineering information systems is through conceptual modeling, which produces a specification of the system to be developed. This specification focuses on what the system should do, that is, on its functionality. Such a specification acts as a prescription for system construction [Roll99b].
In current conceptual modeling approaches, the generated models are represented from the analyst's viewpoint. This can be a drawback because understanding and recording the effect of business changes on requirements has not yet been solved. Requirements also change even as the system is being developed. [Luba93].

Conceptual modeling approaches are currently focused on specifying software functionality aspects, determining what the software should do, and establishing the justifications and restrictions of the software system-to-be. Rolland denominates these activities as the definition of the desired system [Roll99b].

The need to take into account a large number of semantic details in the construction of an information system has led to great diversity in conceptual modeling techniques. One of the most well-founded conceptual modeling techniques is OO-Method [Past96], [Past97], [Pele01]. This is an automatic production process method based on a formal object language called OASIS. This software production environment allows applications to be built in automatic way from conceptual models.

However, we consider that in order to produce software systems that satisfy the user's needs, the conceptual modeling process must be enriched by proposing techniques for understanding organizational processes.

One of the main purposes of this work is to provide systematic guidelines that allow us to obtain a conceptual model for the software system from organizational descriptions. The generated conceptual model must be the input of the OO-Method software production process, which will generate the information system in an automatic way

1.2.3 Proposed solution

As stated above, several research efforts have been made to accurately represent an organizational model (this stage is known as the early requirements phase) [Cast02] [Kolp03] [Bube94] [Cesa02]. In these works, conceptual primitives represent organizational goals, organizational actors, and dependencies among these actors. There are also several research works that focus on the development of requirements models (late requirements) to represent the expected functionality of the information system [Kolp03] [Cock01] [Kula00] [Roll99b].

We consider that the problem of linking organizational models with requirements models in a methodological way has not yet been solved satisfactorily. One of the main reasons for this is the different nature of their specifications. In the early requirements phase, the modeling concepts are associated to the organizational context, while in the late requirements phase, the modeling concepts are associated to the software system to be developed. Therefore, there is a significant conceptual distance between the abstraction levels of the two specifications.

The lack of systematic methods to generate the expected functionality of the software system from the relevant tasks of the organizational model has led to severe limitations in the usefulness of these works in real software development environments.

We propose a methodological approach to reduce the abstraction level of a "pure" organizational model so that it is closer to the requirements model. The reduction process generates a new intermediate organizational model that is correctly adapted in order to systematically generate the requirements model and the conceptual model of the system-to-be. A set of rules for deriving the software specification from the new organizational model is also proposed.

The complete translation process is based on a set of transformational steps that are implemented in a model-driven based approach:

- A goal analysis method is proposed to elicit the current situation of the enterprise. As a result of this step, a "pure" organizational model that reflects the current enterprise situation is generated.
- A goal-based method is proposed to determine which alternatives best satisfy the enterprise goal using a software system.
- A methodological guideline has been developed to reduce the abstraction level between the organizational modeling phase and the system design phase (requirements model and conceptual model). The reduction process is implemented by using a pattern language called FELRE (*From Early*

Requirements to Late Requirements). As a result of this step, an intermediate organizational model (that extends the pure organizational model with monitoring plans and concerned objects) that represents the relevant aspects to be automated is generated. This is done to create a model that is closer to the system-to-be.

- A methodological guideline has been developed to establish the correspondences between an intermediate model and a requirements model. As a result of this step a use-case-based specification is created.
- Finally, a methodological guide has been developed to establish to correspondences between an intermediate model and a conceptual model. An object-oriented model is created as a result of the application of this step.

1.3 Research goals

This thesis has three main research goals:

a) To reduce the abstraction level of a "*pure*" organizational model so that it is closer to the requirements model.

b) To propose a methodological guide that allows a requirements model to be obtained from an organizational model.

c) To propose a methodological guide that allows a conceptual model to be obtained from an organizational model.

The first research goal has been satisfied by dealing with the following sub-goals:

- A goal-based requirements elicitation process, which provides a deep understanding of the organizational environment in order to identify the relevant tasks that should be automated according to their relevance to satisfy the organizational goals.
- A systematic pattern-based process to reduce the abstraction level of a model, by obtaining an intermediate model that is closer to the software system-to-be.

The second and third research goals have been satisfied by dealing with the following sub-goals:

- Extending organization model with monitoring plans and concerned objects in order to create a model that is closer to the system-to-be.
- Developing a methodological guideline that establishes the correspondence between the functionalities that best satisfy the organizational goals and the requirements model.
- Developing a methodological guideline that establishes the correspondence between the functionalities that best satisfy the organizational goals and an object-oriented model.

One of the main advantages of our approach is that it deals not only with *what* or the *how* a piece of software is developed, but also *why*.

1.4 Research environment

This thesis was developed in the context of two well-known research groups: the Object-Oriented Methods for Software Development Group (OO-Method Group) of the Valencia University of Technology (UPV – Universidad Politécnica de Valencia) in close collaboration with the company CARE Technologies S. A., and the Tropos group (Requirements-Driven Development for Agent Software) of the University of Trento, Italy (UNITN).

The work presented here is the result of the efforts of researchers at the OO-Method Group. The results obtained are currently being applied in case studies in both academic and real projects of the Care Technology Company.

There are currently large investments being made to develop tools to incorporate the technology in commercial software development products through R&D contracts between UPV and CARE Technologies.

1.5 Research design

This thesis presents six processes, which are summarized in Figure 1.1. The first five processes occur in two phases: the early and late requirements phases. The last process is related to the validation of the proposed method by developing case studies. The description of each process is briefly explained below.

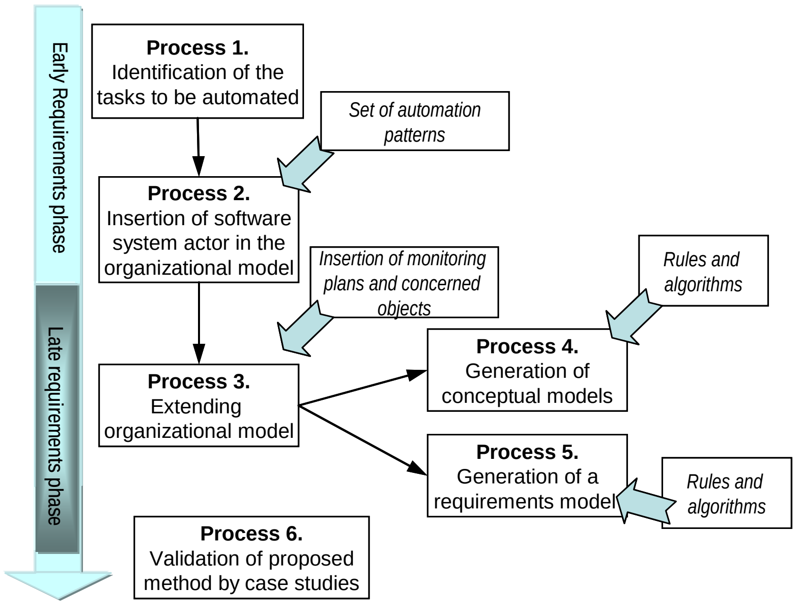

Early Requirements phase
Late requirements phase
Process 1. Identification of the tasks to be automated
Set of automation patterns
Process 2. Insertion of software system actor in the organizational model
Insertion of monitoring plans and concerned objects
Rules and algorithms
Process 3. Extending organizational model
Process 4. Generation of conceptual models
Process 5. Generation of a requirements model
Rules and algorithms
Process 6. Validation of proposed method by case studies

Figure 1.1 Summary of process developed in this thesis

Process 1. Identification of the tasks to be automated.

The starting point of the proposed method is to understand the organizational processes before building an information system. Thus, we started in the early requirements phase, which deals with the analysis of the operational environment where the software system will operate [Yu97]. In this process, a goal-based requirements elicitation process is proposed, which allows us to identify the relevant tasks that must be automated in order to achieve the organizational goal. Chapter 3 describes this process.

Process 2. Insertion of the software system actor in the organizational model.

The strategy of the second process is to insert the software system as an organizational actor into each organizational model. The objective of this process is to consider all the possibilities that exist to delegate tasks and goals from the stakeholders to the software system.
As a result, the system-to-be and its components are represented as a new actor who is responsible for the fulfillment of relevant tasks. We use transformational rules, which are defined by a set of patterns in a pattern language to carry out the equivalence between the organizational and late requirements models. Chapter 4 describes the process.

Process 3. Extending organizational model.

In the third process, the extensions carried out in the organizational models (insertion of monitoring plans and identification of the concerned object) are done to analyze the impact of the system-to-be on the goals of the business. Chapter 5 describes the process.

Process 4. Generation of conceptual models.

In this process, we present the rules and algorithms to establish the correspondence between the elements of the organizational model and the conceptual models of the system-to-be. Chapter 6 describes this process.

Process 5. Generation of a requirements model.

In this process, we present the rules and algorithms to establish the correspondence between the elements of the organizational model and the use case model as well as their corresponding scenarios. Chapter 7 describes this process.

Process 6. Validation of method using case studies.

The last process is related to the validation of the proposed method to obtain the requirements model and the conceptual model. Therefore, three case studies were carried out to evaluate our proposal. Chapter 8 briefly details the case studies.

1.6 Thesis outline

The remainder of this thesis is organized in the following chapters:

Chapter 2. Related works
This Chapter provides a review of the state-of-the-art of some of the relevant topics developed in this thesis. Requirements model generators, conceptual model generators, goal-based requirements analysis methods, and pattern languages proposals. Our intention is to discuss the strengths and weaknesses of each proposal.

Chapter 3. The early requirements
This Chapter presents the goal-based requirements elicitation process that is proposed in this thesis. We detail the process proposed with a set of steps that allow us to find the best way to develop organizational tasks in order to achieve organizational goals. We also briefly describe the basic concepts of the Tropos framework.

Chapter 4. Joining early and late requirements
This Chapter describes the method that is proposed to reduce the abstraction level between the organizational modeling phase and the system design phase. This process is guided by a set of patterns that allows the software system to be inserted into the organizational model as an organizational actor.

Chapter 5 Extending organizational models
This Chapter presents the extension carried out in the organizational model. Therefore, the insertion of monitoring plans and the identification of new elements (called concerned objects) in the organizational model are carried out. The objective of this process is to determine which tasks best fulfill the goals of the business in order to build an information system.

Chapter 6 Linking late requirements with the *ONME* conceptual model
This Chapter describes a method for generating the OO-Method conceptual schema model from the organizational model. It also

introduces the OO-Method approach, describing its four complementary views: Object, Dynamic, Functional and Presentation Models. Then, a brief introduction of the OASIS formal specification language is also explained.

Chapter 7 Linking late requirements with the *ONME* requirements model

This Chapter describes a method for generating the requirements model from the organizational model. This process is conducted using a set of algorithms and rules. This Chapter also describes the concepts of the RETO[1] methodology used in the OO-Method, which is the target of our proposal.

Chapter 8 Cases studies

This Chapter describes the case studies that were carried out as validation of our proposed method.

Chapter 9 Conclusions and further research

This Chapter summarizes the contributions of this thesis, including current and future work and the publications associated with them.

[1] Requirements Engineering TOol

Chapter 2

Related works

This Chapter provides a review of the state-of-the-art of some of the relevant topics in this thesis: requirements model generators, conceptual model generators, goal-based requirements analysis methods, and pattern languages proposals. The objective of this analysis is to adequately contextualize our research work by defining the strengths and weaknesses of the methods analyzed as well as highlighting our contribution with the existing proposals.

2.1 Introduction

Nowadays, several research groups work in developing requirements engineering methods that make feasible the development of information systems which precisely comply with the users needs.

Some of these works are focused on late requirements, which concern the definition of requirements for the system-to-be. Therefore, these proposals consider activities such as requirements analysis [Insf02b] or conceptual modeling [Past99] [Booc99].
Several attempts have been done to produce software specifications from previous stages of organizational modeling. Some of these techniques focus on using requirements as an intermediate model between the organizational model and the software conceptual model [Ort01] [Sant01]. In this approach, the conceptual model, that represent the dynamic and static structure of the system, is viewed as a natural result of the requirements modeling activity that determines the expected functionality of the software system. The advantage of this approach is that it is possible to carry out previous analysis with the requirements specification to include, for instance, non-functional requirements before thinking in generating a conceptual model. Nevertheless, one of the main disadvantages of this approach is the definition of a large number of modeling stages (organizational modeling, requirements model generation, conceptual model generation, implementation generation), which make the software development process costly in time and effort.
Another works focus on generating conceptual models directly from organizational models without going through a requirements model [Alen00]. The main advantage of this approach is the few modeling stages that are needed to derivate a software product. The main disadvantage of this approach is the lack of the appropriate basis to determine the organizational activities that are relevant to be automated by the information system to be developed.
Some of the most relevant works in goal modeling are analyzed that focused on obtaining software requirements from organizational goals. Some relevant works in pattern language technology, which plays a very important role in this thesis, has also been analyzed in this Chapter.

2.2 Methods for requirements model generation

This section discusses five methods that generate requirements models from organizational settings: Santander proposal [Sant02], Ortin proposal [Orti01], Loucopoulos proposal [Louc95], Dijkman proposal [Dijk02] and EKD[1] proposal [Bube98]. The main objective of this analysis is to determine the role of these current methods in the early requirements phase.

Table 2.1 shows an overview of these five methods that considers the following aspects: the inputs of the analyzed methods, theirs role in the development process, the proposed methodology, the methodology used to create the requirements model, and the output of the method. It is important to point out that this is not an exhaustive analysis and it only pretends to highlight some similarities and differences between the methods.

1 EKD Enterprise Knowledge Development

Table 2.1 Overview of methods generating a requirements model

	Santander proposal (2002)	**Ortin Proposal (2001)**	**Loucopoulos proposal (1995)**	**Dijkman proposal (2002)**	**EKD proposal (1995)**
Input of the method	Business models (early requirements phase)	UML Diagrams (Role diagram, sequence diagram and process diagram)	Business goals	Business process (activity diagram)	Analysis and understanding of the enterprise
Notation	i* framework	UML diagrams	Teleological Views	UML diagrams	EKD Notation
Role in the development process	It uses guidelines to find use cases of the software system to-be; it also requires the experience of the requirement engineers.	It uses role diagram and sequence diagram to find use cases of the software system to-be.	It uses models to show scenarios with the different situations that satisfy the vision and criteria for changing the business.	It uses a procedure to transform business process models into UML use case diagrams.	It uses multiple and complementary views for modeling process in an enterprise.
Methodological approach	This method is focused on analyzing business goals in order to obtain use cases.	This method is focused on determining functionalities of the software system to-be.	This method is focused on capturing the reason that exists behind the business tasks and to detail how a certain activity has been assigned to an organizational actor.	This method is focused on creating meta-models for both use cases and business process in order to compare them and detect differences and similarities.	Computer aided documentation of knowledge about enterprises
Method to define requirements model	Guidelines and heuristics are provided	Some steps are provided	Three complementary views to carry on the analysis are required.	A table of mapping of primitives is provided and a set of steps is proposed	Analysis of the information of the proposed models
Output of the method	Use case models and scenarios represented in UML	Use case models and scenarios represented in UML	A requirement model for the system-to-be	Use case model represented in UML	A set of high level requirements for the information system to- be

A brief description of each proposal is presented below. The description makes emphasis on the advantages and disadvantages of each method.

2.2.1 The Santander proposal (2002)

The main objective of the Santander proposal [Sant02] is to generate scenarios and use cases represented with the UML for the software system from organizational models represented with the *i** framework.

The author argues that the *i** framework provides an early understanding of the organizational relationship in the business domain, which is needed to develop a software system that complies with the users needs. This is because the organizational modeling

process requires an integrated view of the functional and non-functional aspects, which are needed to support the alternative selection of early requirements.
Santander proposes a set of heuristics that helps the requirements engineer to determine the existence of potential use cases from the organizational model specification.
The use case model was adopted by Santander as a first step to describe the functional requirements of the software system. Usually, the UML use cases are developed without considering the organizational requirements. In Santander works, it was argued that use cases developed from the organizational model permit the requirements engineer to establish relationships between the functional requirements of the system and the organizational goals previously defined in the organizational model.
The steps to integrate an *i** organizational model and a UML use case model according to Santander proposal are shown in Figure 2.1,

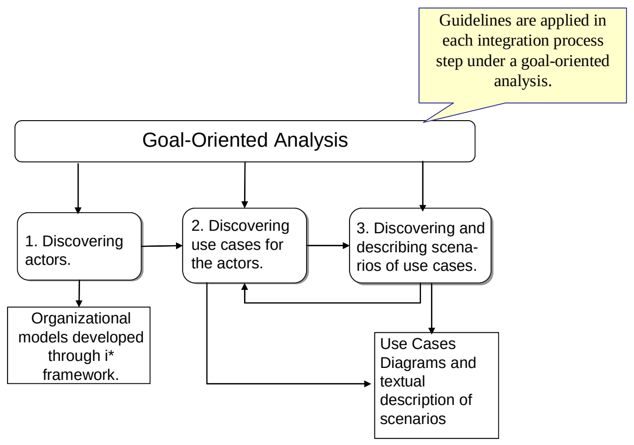

Guidelines are applied in each integration process step under a goal-oriented analysis.
Goal-Oriented Analysis
1. Discovering actors.
2. Discovering use cases for the actors.
3. Discovering and describing scena-rios of use cases.
Organizational models developed through i* framework.
Use Cases Diagrams and textual description of scenarios

Figure 2.1 Steps to integrate *i**organizational model and the UML use case models [Sant01]

1st Step: Discovering actors. The inputs of this modeling stage are the *strategic dependency model (SD)* and the *strategic rationale model (SR)*, which reflect the business behavior. This step analyzes the relevance of each organizational actor according to the informational system-to-be. The author proposed some guidelines to support the process to discover the actors.

Guideline G'1: all actors in *i** must be analyzed for a possible mapping into actors of the use case models;

Guideline G'2: if the actor in *i** model is external to the intended computational system, then it is a candidate to be transformed into a use case actor;

Guideline G'3*:* if the actor has some kind of dependency with the actor who represents the system to be developed, then it is a candidate to be transformed into a use case actor;

Guideline G'4*:* IS-A relationships in *i** model are mapped as generalizations links in the UML use case diagrams;

2nd Step: Discovering use cases. An analysis of each actor is carried out in order to determine its role in the dependency relationships. The role of each actor is also analyzed in order to select those actors who play the role of *dependee* in the dependency relationship[1]. The analysis of dependencies is carried out as follows:

Guideline G'5: for each actor in the model, we must analyze all the dependencies of the analyzed actor with the actor that represents the system to be developed. The objective of this guideline is to discover the use cases from the actors.

Guideline G'6**:** for each actor in the model, we must analyze all the dependencies of the actor that represents the system-to-be with the organizational actors. The objective of this guideline is to discover new use cases from relationships of this kind.

Guideline G'7: classify each use case according to its objective type (contextual objective, user objective, sub-function objective).

3rd. Step: Discovering and describing the main and alternative flows of the use cases: The primary and secondary scenarios are described in this phase, as well as the relationship between use cases. This information is taken from the strategic rationale model. As a result of this step, the use case diagram and the textual scenario description for each use case is generated. The guidelines to support this step are the following:

1 This concepts are detailed in Chapter 3 of this thesis (subsection 3.3.2.1)

Guideline G’8: analyze each actor and its relationships in the strategic rationale model in order to extract information that generates the description of the main and alternative flows.
Guideline G’9: each use case must be analyzed to check for the possibility to refine it and generate new use cases.
Guideline G’10: create the use case diagram using the discovered use cases and actor.
The main contribution of this method is the set of heuristics proposed, which helps the requirements engineers to develop the UML use cases based on the organizational models. This method also represents the software system as an actor in the organizational models in order to determine the activities of the organizational model needed to be automated.
One of the main issues of the Santander proposal is that the heuristics presented are not enough to obtain use cases in a systematic manner. For example, the heuristic presented in second step, which has the objective of discovering the potential use cases, does not clearly suggest how to obtain them. Sometimes, the uses case will be the resultant product of an analysis of task and resources dependencies, sometimes, the use cases need to be obtained directly from goal dependencies, or even from resources dependencies. A similar problem occurs with the use cases scenarios determination, since it is necessary to analyze the four organizational models (dependency and strategic rationale with or without the software system actor) to obtain the scenarios.
The proposed heuristics are just guidelines to support the integration of *i** with use cases modeling techniques. It is pointed out that for the application of these heuristics; great experience is required from the requirements analysts.

2.2.2 The Ortin proposal (2001)

The work proposed by Ortin [Orti01] presents a strategy to systematically obtain use cases models and conceptual models from a organizational model specification.
Figure 2.2 shows a schema of this proposal, which is based on UML activity diagrams.

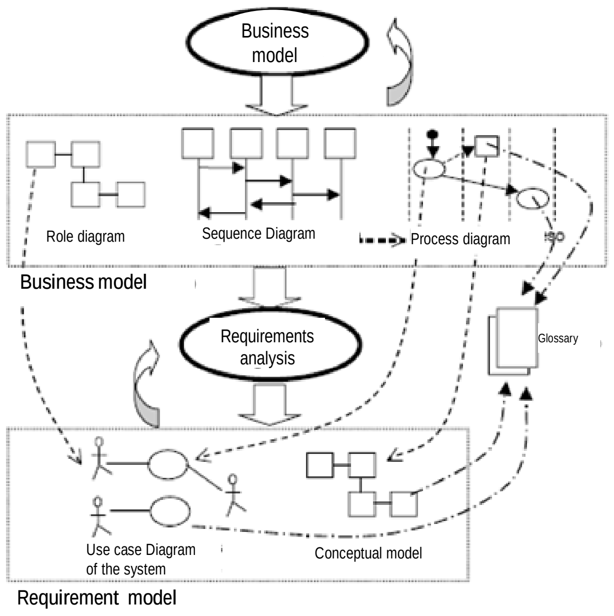


Figure 2.2 Traceability relationships among an organizational model and a requirements model

The authors argue that the organizational model can be the most important basis to the requirements specifications of an information system, which pretends providing support to the enterprise activities.
Therefore, use case modeling and conceptual modeling are carried out at the same time in this proposal, making it easier the identification and specification of the suitable use cases, according to the suggested by Korson [Kors99].
The organizational modeling activity is implemented using the traditional UML diagrams: business use cases diagram, roles diagram, sequence diagram and process diagram, as well as a glossary which contains the business rules. The initial use case collection and the preliminary conceptual model are obtained from these UML models.

In the Ortin proposal, it is necessary to generate a use cases system for each activity diagram in order to generate the use cases model for the information system.
In the example shown in Figure 2.3, the following business activities were considered as potential use cases: fill order, send order, notify accepted order, notify rejected order, analyze viability, order fabrication, and organize production. It is necessary to point out that some of the use cases will not be obtained from the process diagram, but they will be detected by describing the identified use cases and by acquiring a great knowledge about the requirements that should be supported by the information system.
A specific template must be used to describe the detected use cases. Once the use cases have been detailed, these are connected with the specification in the glossary in order to make the correspondence between business use cases and system use cases.
The main contribution of the Ortin method is the systematic transition from the business modeling to the requirements modeling phase and the conceptual modeling phase.

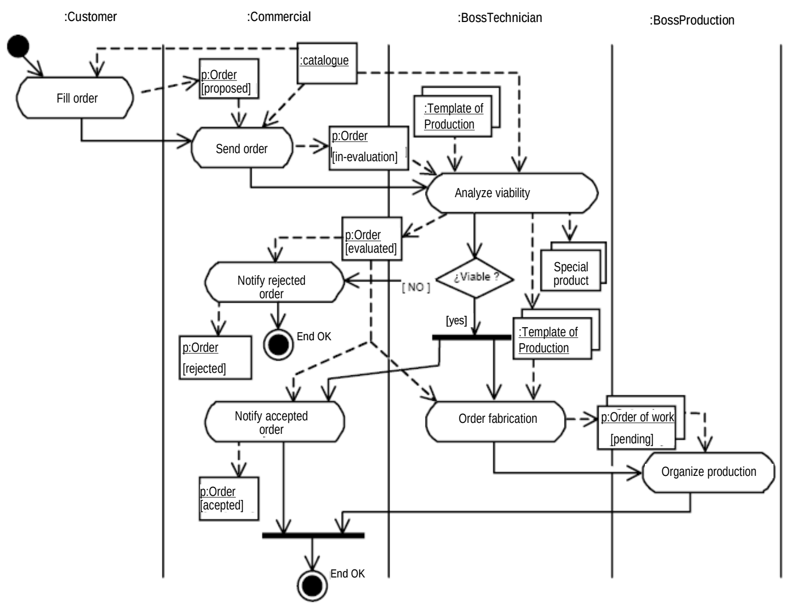


Figure 2.3 Process diagram which enables to obtain the use cases.

One of the main issues of this proposal is that it is focused on the information system generation, and most of the key aspects of the business modeling have been neglected, such as the relationships of tasks with organizational goals, the intentionality behind the tasks, the description of types of dependencies which join the actor, the strength of this dependency and the task decomposition.
We argue that most of the issues of this proposal have the source in the weakness of UML to express the complex behaviors that exist in enterprises [Cesa02] [Alen03].

2.2.3 Loucopoulos proposal (1995)

This proposal is based on the explicit modeling of the organizational objectives, the social roles and the operations from the Teleological point of view. One of the main premises of this proposal is that an organizational model is relevant if it allows us to provide explanations about the behavior of the enterprise. Teleological proposal establishes the analysis of goals and the analysis of organizational dependencies as the first step for and in-depth understanding of the enterprise. This approach, which has been called teleological, is useful for capturing the reasons that exist behind the business task and also for explaining how a certain activity has been assigned to a specific organizational actor.

The teleological technique is composed of five basic elements: goals, roles, actors, processes and resources. The goals are the core of the modeling process because they provide clear explanations about the current and future configuration of the enterprise. The concept of actor considers people as organizational units and as basis constructs. Processes are the mechanisms that permit changes of states in the organizational system. Finally, resources are the informational or physical means that are produced as result of the business processes.

Teleological approach includes three complementary views for representing an organizational model: the teleological, social, and process views. Each phase is described below:

Teleological view (Figure 2.4): The goals of the stakeholders are represented in this view.

The goals imply intentions and also represent solutions to the problems of the enterprise.

The constraints, which are operational goals, must be formulated in terms of precise properties and actions

Social view: the organizational actors and their interactions are detailed in this view (Figure 2.5). The actor is a key modeling factor since the actor is the entity responsible for executing the organizational activities. An actor can be and individual (person, a software system, etc) or an organizational unit (department, division, section, etc). The roles are a set of processes that are assigned to a specific agent. This assignation is dependent on their goals and capabilities. An actor can play several roles at the same time.

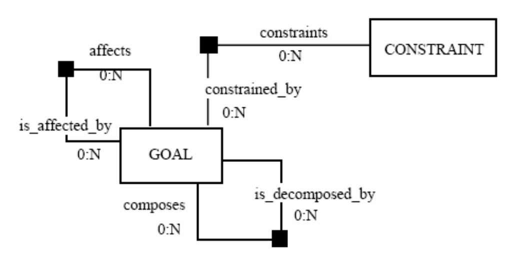


Figure 2.4 The teleological view of the enterprise modeling

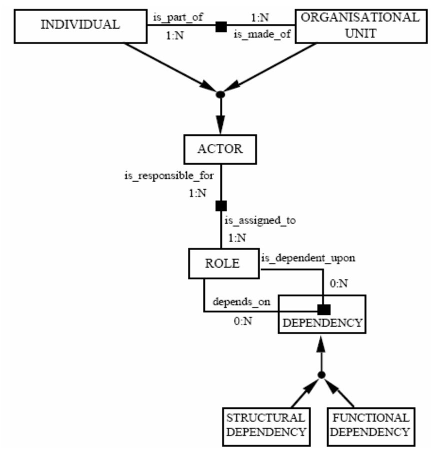


Figure 2.5 Meta-model of the social view

Process view: this provides a general view of the current process in the enterprise (Figure 2.6). This view also considers the resources that are relevant for the execution of the processes. The process view permits the representation of triggers that correspond to changes in the business. The events represent the dynamic dependencies among the processes. The events can be generated by processes or by temporal conditions.

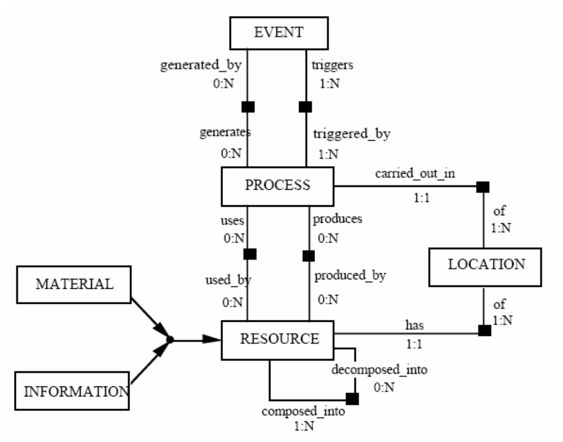


Figure 2.6 Meta-model of the process view

Some of the advantages of this proposal are:
The views of the teleological model can be very useful for constructing an initial set of requirements for either the business model as for the software system.
The proposal considers a well-defined graphical notation for each business view. The views consider only a small number of modeling elements.
The technique enables us to define functional and structural dependencies. This characteristic is useful for determining when the tasks of a certain actor influence the execution of tasks of other organizational actors.
However some disadvantages of this proposal are:
Two kinds of analyses must be carried out. The first is the determination of the high-level objectives of the enterprise and their refinement until the operational activities are elicited (prescriptive analysis). The second analysis concerns the details of the operations of the current business processes (descriptive analysis). However, no details are given in order to reconcile the two specifications when there is no precise match between them.
There is only a brief explanation about goal decomposition. No details are given about conflicting or redundant goals. Also, there is

no formal description of the elicited goals, which makes it difficult to validate the goal model.
Only a brief explanation of the traceability among the different views of the proposal is given.
There is not an explicit association between the goal model and the process model. This makes it difficult to identify the processes that give support to a specific enterprise goal.
The complete explanation of the business model implies the analysis of the three models. Therefore, it is not possible to have a unique global view of the current business process, which can be very useful for business process reengineering.

2.2.4 The Dijkman proposal (2002)

Dijkman et al [Dijk02] propose a technique to derive functional requirements, specified with use case diagrams, from existing business process models.
The authors argue that, due to the existing similarities between the definitions of business process and use cases, a business processes can also be described by using use case models [Nurc98] [Jaco99] [Jaco94] [OMG01]. To validate this assumption, Dijkman proposes the creation of meta-models for both use cases models and business process model in order to compare them and detect differences and similarities.
The comparative analysis between both meta-models results in a mapping which is the basis for the transformation procedure of business process models into use case diagrams.
The meta-model of a use case diagram is shown in Figure 2.7. It is a simplified version of the meta-model that can be found in the UML specification [OMG01].

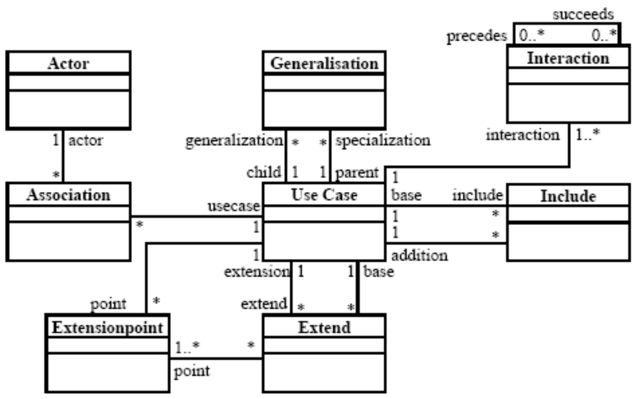


Figure 2.7 Use case meta-model

The business process meta-model is shown in Figure 2.8. It has been constructed by generalizing the meta-models of 18 business tools analyzed in [Domm99].

The basic steps to define the mapping between business processes and the use case models are the following:

First, an initial mapping between concepts and relations of both, business and uses case specifications must be performed. The initial mapping is based on the definitions of the concepts and relations. A summary of the mapping correspondences for business process modeling concepts and use case modeling concepts is shown in Table 2.2.

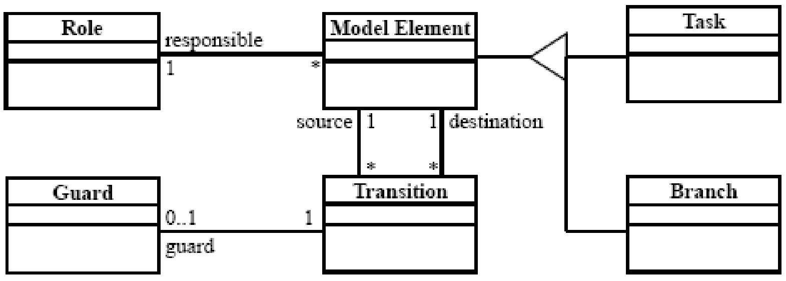


Figure 2.8 Business process meta-model

Then, a formal specification is proposed in order to verify the correctness of the initial mapping among models. The formal specification of the mappings specifies the extension of the concepts and relations. Thus, the validations of the mapping consists of a Z specification of the mapping itself, and a *Z* specification that we derived from the meta-models on the modeling techniques.
Nevertheless, the Dijkman approach has as a limitation: it does not provide absolute proof that the procedure is correct. The reason for this is that the formal specification of the procedure that is used as proof merely allows us to validate the procedure. It cannot provide a formal proof of the correctness of the procedure [Dijk02].

Table 2.2 Mapping from business process to use case concepts

Business Process Concept	Use Case Concept
Role	Actor
Step	Use Case
Association between Role and Step	Association between Actor and Use Case
Task	Interaction
Task in a Step	Interaction in a Use Case
Transition between Tasks in the same Step	Ordering between Interactions in the same Use Case
Guard on Transition	Constraint on Interaction
Alternative Path through a Branch	Alternative Path Description of a Use Case, or Extending Use Case

2.2.5 EKD proposal (1995)

EKD [Kiri94] [Bube94]is an approach that provides a systematic and controlled way of analyzing, understanding, developing and documenting an enterprise and its components, by using Enterprise Modeling.

The Enterprise Model contains six interrelated sub-models (Figure 2.9). Each of them represents some aspect of the enterprise. The types of sub-models and issues are:

Goal Model (GM) focuses on describing the goals of the enterprise. This model permits the identification of relevant properties of the goals such as criticism, priority, relationships, and relevance.

Business Rules Model (BRM) is used to represent the set of restrictions that affect the satisfaction of a specific goal of the goal model.

Concept Model (CM) is used to strictly define the "things" and "phenomena" one is talking about in the other models. It represents enterprise entities, attributes, and relationships. Entities are used to define stricter expressions in the Goals Model as well as the content of information sets in the Business Processes Model.

Business Process Model (BPM) is used to define enterprise processes, the way they interact and the way they handle information as well as material.

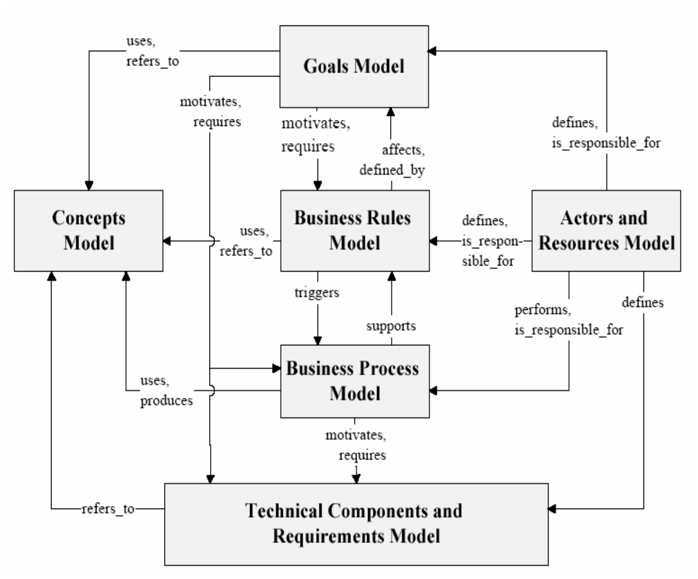


Figure 2.9 The sub-models comprising the enterprise model

Actors and Resource Model (ARM) is used to describe how different actors and resources are related to each other and how they are related to components of the Goals Model, and to components of the Business Processes Model.

The Technical Components and Requirements Model (TCRM) becomes relevant when the purpose of EKD is to aid defining requirements for the development of an information system.

The focus on EKD modeling is placed on the definition of the technical system that is needed to support the goals, processes, and actors of the enterprise. Initially, a starting set of high level requirements for the information system as a whole are elicited.

Later, based on this information, the analyst must structure the information system in a number of subsystems, or technical components. TCRM is an initial attempt to define the overall structure and properties of the information system to support the business activities, as defined in the *BPM*.

When the objective is to develop an information system to support the processes, there is a need to deal with technical information system requirements, initially in a less formal way. Therefore, the EKD approach includes a simple sub-model to describe, and to relate to each other, initial, and unclear information system requirements.

This sub-model resembles goals and information system models as a whole. Initially one needs to develop a set of high level requirements or goals, for the information system as a whole. Based on these, it is necessary to structure the information system in a number of subsystems or technical components.

For each subsystem, a set of goals (which are more specific) and requirements are defined. These goals and requirements have to be derived from, and be consistent with, the earlier sub-models discussed above. The Technical Components and Requirements Model is an initial attempt to define the overall structure and properties of the information system to support the business activities, as defined in the Business Processes Model.

Some of the advantages of this proposal are:

The EKD approach, which is based on multiple and complementary views, approaches the modeling process in an incremental way.

There are well-defined graphical notations for each one of the views that makes up the business model.
However, some disadvantages of this proposal can be summarized as:
The semantics of the organizational model must be represented using a large number of models, which makes the practical application of this proposal difficult.
There is not a well-defined method that allows us to derive the general goals of the enterprise from the operational goals of the stakeholders. Only a description of each sub-model is presented in the proposal.
It is not possible to establish the required efforts to produce an automatic transformational process between the business model and the requirements model since neither the method nor the heuristic are described in the proposal to map these models.

2.3 Methods for conceptual models generation

Table 2.3 shows an overview of two methods [Ort01] [Alen00] that generates object-oriented conceptual models from organizational models. The table considers the following aspects: the inputs of the proposed methods, their role in the development process, the methodology proposed, the methodology to generate the requirements model, and the outputs of the method. This is not an exhaustive analysis and it only pretends to highlight some similarities and differences between the methods.

Table 2.3 Overview of methods to generate a Conceptual Model

	Alencar proposal (2003)	Ortin Proposal (2001)
Input of the method	Organizational models (the early requirements phase)	The UML Diagrams (process diagram)
Notation	The *i** framework	UML diagrams
Role in the development process	Guidelines are used for the proposed method to find a business class diagram	Role and sequence diagrams are used for the proposed method to find the use case model for the software system.
Methodological approach	This method focused on obtain a business conceptual model	This method focused on the functionality of the software system
Method to define requirements model	Some heuristics are provided	Some steps are provided
Output of the method	Conceptual model represented in UML	Conceptual model represented in UML

Following, a brief description of these proposals is presented where we have put emphasis on the advantages and disadvantages of each method.

2.3.1 The Alencar proposal (2003)

In Alencar proposal [Alen03], a transformational process to derive late requirements specifications specified in pUML (precise Unified Modeling Language) from late requirements model represented in *i** framework is proposed. The object constraint language (OCL) is used to cover the lack of pUML to represent invariant specification restrictions, preconditions etc., which are necessary to correctly represent the behavior of the information system in the conceptual model specification.

According to Alencar, the UML is suitable to capture requirements in the late requirements phase (software product specification) although it is generally focused on the complexity, consistency and automatic verification of the functional requirements [Booc99]. However the UML is badly-equipped to capture the requirements in an early phase (model organizational specification). Thus, it does not provide answers to the following questions: How does the software system help to accomplish the organizational goal? Why is the system necessary? Which alternatives were considered and how the stockholders interests are oriented to? In Alencar works, the *i** framework was chosen because it permits to answer those questions, also it permits to represent alternative solutions and it also offers modeling concepts such as goals and soft goals [Mylo99].

The guidelines proposed by the author for the generation of class diagrams were originally proposed in [Alenc99] [Cast01]. Later on, these guidelines were extended to support the structuring elements of *i** [Alenc03].

Bellow, a short description of the guidelines is presented:

Guideline G1: Related with the mapping of *i** actors.

Guideline G1.1: *i** actors (agents, roles or positions) can be mapped to UML classes;

Guideline G1.2: *i** relationship IS-PART-OF between actors can be mapped as a class aggregation in UML;

Guideline G1.3: *i** relationship IS-A between actors can be mapped to class generalization /specialization in UML;

Guideline G1.4: *i** relationship OCCUPIES between an agent and a position can be mapped as a class association in UML named OCCUPIES;

Guideline G1.5: The *i** relationship COVERS between a position and a role can be mapped as a respective class association in UML named COVERS;

Guideline G1.6: The *i** relationship PLAYS between an agent and a role can be mapped as a respective class association in UML named PLAYS;

Guideline G2: Related with the mapping of *i** tasks.

Guideline G2.1: A task defined in SD (Strategic Dependency) model can be mapped as an operation in the interface that is done by the class that represents the *dependee*. The name of the newly created interface is constituted by the names of the classes that represent the *dependee* and the *depender*.
Guideline line G2.2: A task defined in the SR (Strategic Rationale) model can be mapped as an operation with private visibility in the class that represents the actor which the task belongs to;
Guideline G3: The *i** resources can be mapped to UML classes.
A resource can be mapped as a class in UML if this dependence has the characteristics of an object, or as an attribute with private (SR model) or public (SD model) visibility.
Guideline G4/G5: Related with the mapping of *i** goals/*softgoal*s.
Goals can be mapped as boolean (goals) or numeric (*softgoal*s) attributes with private (SR model) or public (SD model) visibility. An association is created between the depender class and the dependee class.
Guideline G6: Related with the mapping of *i** relationship task decomposition, this are represented by pre and post conditions (expressed in OCL) of the corresponding UML operation.
Guideline G7: Related with the mapping of *i* means-end* relationship. This is used to generate disjunctions (expressed in OCL) of all possible means achieving the end.
The result of the early requirements phase is a class diagram (in which the classes have attributes and methods). It is important to point out that not all the concepts captured in early phase have a correspondence with modeling concepts in the conceptual model of the software system. In this sense, some elements of the organizational model do not have a counterpart in the software system model. This is because some of the organizational activities that must be performed manually do not need to be represented in the software system model. On the other hand, many elements represented in a software model concerns technical details that are out of the scope of an organizational model.
In this proposal, guidelines that contribute to the formalization process of requirements expressed through the technique *i** with

MAL language (Modal Actions Logic) are presented. These guidelines allow relationships to be established between the fragments of the formal specification in MAL and other organizational goals described in the *i** models.
The main contribution of this method is that it provides guidelines to obtain a classes diagram from the elements of the organizational model specified in *i**. This framework allows expressing the reasons ("*why*") of the processes (motivations, intentions and reasoning) [Yu 98] that exist behind the activities in a organizational model. Furthermore, in this study guidelines are provided for the formalization of requirements in MAL language.
However, in this proposal, only the correspondence between the elements of the organizational model and the conceptual model are analyzed (class diagrams). As a result, when using the guidelines demonstrated in this study, what is constructed is the conceptual model of the organizational model, and not the conceptual model of the information system. If this conceptual model is implemented the information system is not generated, but what is generated is a software system that allows animating the organizational model.
In order to derive the conceptual model (of the information system) from the organizational model, it is necessary to implement a previous stage in which the activities of each actor, that need to be automated, must be determined.
A great deal of experience is needed from the analyst's side to carry out the correspondence in the models.

2.3.2 The Ortin proposal (2001)

This proposal presents a method to obtain conceptual models from business models represented using the UML activity diagrams.
The initial conceptual model generated is obtained by exploring the specification of the business the use case model, which represents the domain data.
In Figure 2.3 a process diagram is shown, where the information objects are represented as rectangles. These information objects may be considered as concepts (in design stage these objects will produce classes, as long as the software system requires to give support to

such concepts). Figure 2.10 shows the initial model that can be obtained from the process diagram shown in Figure 2.3. Finally, this initial model can be refined in order to obtain attributes, relationships with other classes and restrictions of each object.

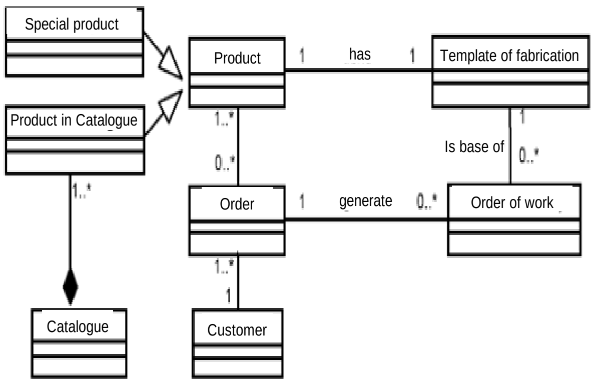


Figure 2.10 Example of the initial conceptual model

One of the main contributions of this model (as mentioned in section 2.1.2) is the generation of requirements and conceptual models. This generation is carried out in parallel, which makes easier the identification and specification of the most appropriate use cases.
However, one of the main weaknesses is the lack of guidelines to map business and conceptual models; therefore this process is responsibility of the analysts.

2.4 Current Methods for Goal-based requirements analysis

Requirements Engineering (RE) has been described as "the branch of systems engineering concerned with the real-world goals for, functions of, and constraints on software-intensive systems. It is also concerned with how these factors are taken into account during the implementation and maintenance of the system, from software specifications and architectures up to final test cases" [RE02]. Thus, requirements engineering already assumes the view that "*real-world* *goals, functions and constraints*" are the source of the requirements for software systems. Goal-Directed Requirements Engineering (GDRE) is a branch of RE, which is concerned with the definition of methods for defining the complete requirements for a software system starting from goals stated by stakeholders.

GDRE methods generally define a Goal to be a [Mylo01] "*condition or state of affairs in the world that the stakeholders would like to achieve.*" Several GDRE methods have been developed in the last years. Examples of such methods are KAOS (Knowledge Acquisition in an automated Specification) Lams95] [Lams00], Goal-Based Requirements Analysis Method GBRAM [Anto97] [Anto98], ESPRIT CREWS [Roll98a] [Roll99c], and NFR [Mylo99] [Chun00].

More recently, the Tropos framework has been proposed as a basis for UML extensions for agent-oriented software development [Mylo01] [Gior02].

2.4.1 The GBRAM proposal (1996)

In GBRAM (Goal-Based Requirements Analysis Method), several principles are assumed for identifying and refining goals into operational requirements. First, the process of acquiring requirements involves an integrative approach, focusing on both abstract goals and concrete behaviors that stakeholders expect the system to exhibit.

GBRAM assumes that goals have not been previously documented or explicitly elicited from stakeholders and that analysts must work from various sources of available information, each with its own scope of knowledge, to determine the goals of the desired system. It also supports the elaboration of goals to represent the desired system.

A detailed presentation of how to apply the method from the initial identification of goals to the translation of those goals into operational requirements is available in [Anto97]. Following, we provide a brief overview of the method, differentiating between the goal analysis and goal refinement activities. Goal analysis concerns with the exploration of available information sources for goal identification followed by the organization and classification of goals. Goal refinement concerns with the evolution of goals from the stage they are first identified, to the stage where they are translated into operational requirements for the system specification. The goal analysis activities may be summarized as follows:

- Exploration activities entail the examination of the inputs.
- Identification activities entail extracting goals and their responsible agents from the available documentation.
- Organization activities involve the classification of goals and the organization of those goals according to goal dependency relations.

The goal *refinement* activities may be summarized as follows:

- Refinement activities entail the actual pruning of the goal set.
- 'Elaborate' refers to the process of analyzing the goal set by considering possible goal obstacles and constructing scenarios to uncover hidden goals and requirements.
- 'Operationalize' refers to translating goals into operational requirements for the final requirements specification.

Figure 2.11 shows the activities which an analyst is involved with when applying the GBRAM.

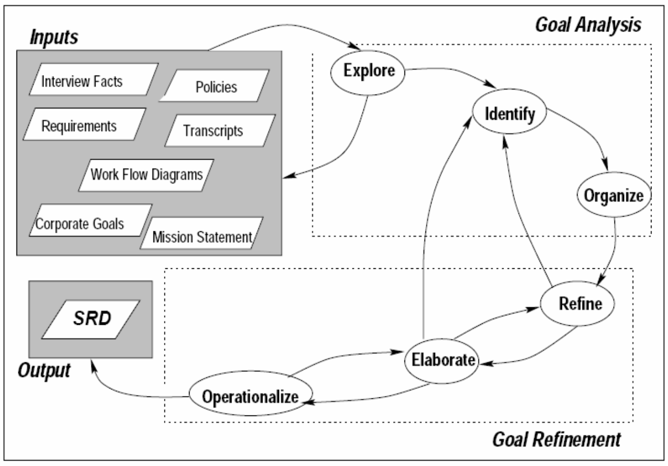


Figure 2.11 GBRAM modeling activities

One of the main contributions of this work is the definition of a clear method to elicit the abstract goals in order to define a set of operational goals which will lead to the requirements for the information system. This approach makes possible the definition of the reason of the existence of each one of the business activities. Also GBRAM offers appropriate mechanisms to detect redundant goals, and also for consolidating equivalent goals.
The goal restrictions are used as “finishing” mechanisms. This is useful for the analyst to determine when the goal refinement process must finish. This proposal considers the definition of pre and post conditions needed for goal fulfillment.
On the other hand, some disadvantages in this proposal are: this proposal has not considered the interaction between goals and quantitative non-functional requirements, such as performance and reliability. In this case, improvement and maintenance goals do not become operational directly as achievement goals. There is no a formalization of the elicited goals. Therefore, the description of the goals is made in natural language. This is a disadvantage because natural language cannot be used to perform formal verifications or reasoning about the elicited goals.
This approach does not propose a graphical notation for the proposed goal category. Therefore, the only unique material available to analysts is the natural language goal definition. The modeling process of GBRAM ends when the operational goals have been elicited. Therefore, this technique does not offer mechanisms to define a business model that explicitly associates the business process model with the elicited goal model.

2.4.2 KAOS proposal (1993)

KAOS is a formal approach for analyzing goals and producing requirements based on pre-stated goals. There is abundant KAOS literature, e.g., [Dard93], [Lams95], [Lams98], [Dari96], [Lams00], [Lams01]. KAOS approach is mainly oriented towards ensure that high-level goals identified by stakeholders, fulfill system requirements. The method is composed of the following components:

KAOS is a specification language based on concepts such as: Object action, agent, goal, constraint, etc. This language uses real-time temporal logic to represent constraints on past and future states.

KAOS proposes an elaboration method for transforming stakeholder goals into requirements for the software system. This method includes classical questions, such as how and why, to refine and abstract goals in the goal-reduction graph: the identification of pre, post and trigger conditions of goals, the identification of agents to which goals are to be ascribed, identification and resolution of conflicts, etc.

Following, the main steps for the method for requirements elicitation are presented:

Step 1. Identifying goals from initial documents.
Step 2. Formalizing goals and identifying objects.
Step 3. Eliciting new goals through WHY questions.
Step 4. Eliciting new goals through HOW questions.
Step 5. Deriving agent interfaces.
Step 6. Identifying operations.
Step 7. Operationalizing goals.
Step 8. Anticipating obstacles.
Step 9. Handling conflicts

A meta-level knowledge base is used for guiding decisions during the elaboration process. This meta-level knowledge base contains:

- A classification of goals
- Rules to ensure the consistency and completeness of requirements.
- Tactics and heuristics for driving the elaboration and selecting among alternative goals

Some advantages of the KAOS approach are:

KAOS classifies goals into: achieve, cease, maintain, avoid and optimize goals. Achieve and cease goals are said to generate behaviors. Maintain and avoid goals are said to restrict behaviors. Optimize goals are said to compare behaviors [Dard93]. This classification enables the analyst to capture the complex organizational setting.

A specific method to face each modeling stage is presented in KAOS. This constitutes one of the main kindnesses of this technique; since precise guidelines are provided to build the modeling diagrams, and all the elements of the KAOS meta-model have formalization in temporary logic.

On the other hand, the disadvantages of KAOS, from our point of view, can be summarized as: the KAOS literature does not explain the need to classify goals in this way. KAOS uses domain knowledge that is considered as objective knowledge, to reduce goals into sub-goals [Dard93], [Dari96]. Also, KAOS does not encourage the challenging of goals given, expressed by stakeholders, with the exception of conflict resolution [Lams98]. KAOS provides tools for transforming stakeholders' goals into requirements, but without making sure that these are the right goals to define the requirements on.

2.4.3 Tropos proposal (2005)

Tropos presents a formal framework for reasoning with goal models. In particular, the Giorgini research works [Gior05] introduce a qualitative and an axiomatic numerical for goal modeling primitives. Also, label propagation algorithms are shown to be sound and complete according to their respective axioms.

The work of Giorgini has been done in the context of the Tropos methodology, which adopted the *i** modeling framework [Yu95]. The *i** framework views organizational models as networks of social actors that have freedom of action, and *depend* on each other to achieve their objectives and goals, carry out their tasks, and obtain needed resources.

Tropos approach is a modeling framework for goals which includes AND/OR relationships among goals, but also allow more qualitative goal relationships, as well as contradictory situations [Bohe96] [Lams98]. The analysis of contradictory situations is carried out by introducing goal relationships labeled "+" and "-", that models respectively, a situation where a goal contributes positively or negatively towards the satisfaction of another goal.

The main advantage of this approach is the use of quantification to evaluate the degree of goal accomplishment. This characteristic enables the analysts to evaluate different alternatives to satisfy the enterprise goals with the highest probability of success.
Also, this approach offers a well-founded set of axioms for defining goal relationships. This proposal also provides axioms to lead the qualitative and quantitative reasoning with goal models.
Additionally, the proposed approach introduces a well-defined goal relationship to indicate positive and negative contributions of the satisfaction of a goal into the satisfactions of other goals in the model.
On the other hand, the main issue of Giorgini works is the lack of mechanisms to associate the goal structure generated by the application of his technique with the strategic models of the Tropos framework. This is a consequence of the modeling strategy of this approach, where the focus is placed on the analysis of the goals in the abstract, without considering the specific actors that are responsible for the elicited goals. Therefore, for novel analyst in Tropos it could be complicated to take design decisions to assign a certain goal to a specific actor in the enterprise.

2.5 Pattern language proposals

One of sources for the pattern approach has been given by Christopher Alexander in the book *“The timeliness Way of Building*” [Alex79] about the urban and building construction. This book offered the particular vision of the author about the recurrent problems that used to exist in the architecture of towns and cities, and, in general, in any kind of building. Alexander described these problems and their solutions using the term “pattern”.
Each pattern describes a recurrent problem that occurs in a specific environment. The pattern describes the context where the problem could be found and also offers a solution for the presented problem.
The pattern describes the environment of the solution to the problem, in such a way that this solution can be used more than a million of times without doing it the same way twice [Alex77].

After examining the Alexander works, several research groups observed the same situation in software development, where there is clear evidence of the patterns in all design levels, from high level architectures to detailed design problems. Although the pattern approaches were defined, mainly, in low abstractions levels (implementation and design) at the present time, their evolution has extended to almost all areas related to software development. Getting into more specific aspects, the patterns are generally classified based on its abstraction level [Garz02].

Bushmann et al (1998) define three levels of abstraction in defining patterns:

- An **Architecture Pattern** expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them. [Busc98].
- A **Design Pattern** provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly recurring structure of communicating components that solves a general design problem within a particular context [Gamm95]).
- An **Idiom** is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language [Copl91].

Although this classification is the best known in computer science field, it represents only a subset of the possible types of patterns. For example, in [Rieh96] one division of conceptual and analysis patterns, is done. In [Fowl97] design and implementation patterns are also analyzed. Other kinds of patterns are those used in the agent-oriented approaches, which have been used to design multiple aspects of a system.

Some examples of agent-based methodologies that include the use of patterns are Tropos [Gior05], Kendall’s methodology and PASSI. Kolp et al. present a set of patterns in [Kolp01] as part of the Tropos

methodology, which uses patterns (called *styles*) to describe the general architecture of a system under construction.
Kendall [Kend99] also includes a catalogue of patterns as a part of a technique to analyze and design agent-based systems. The patterns in that catalogue are more general than those presented in our work, since they include, not only interactions, but also the roles themselves (it should be noted that the concept of *role* there, comes from role theory and it is not identical to the concept used here). Cossentino et al. present in [Coss02] the design of a particular type of agent pattern immersed in the PASSI methodology. They define a pattern consisting of a model and an implementation code. The model includes two parts: structure and behavior. Structural patterns are classified into: action patterns, which represent the functionality of the system; behavior patterns, which can be viewed as a collection of actions; component patterns, which encompass the structure of an agent and its tasks; and service patterns, which describe the collaboration between two or more agents. Implementation code is available for two agent platforms, named, JADE and FIPA-OS.

2.6 Summary

In this Chapter, several proposals in research fields close to the research work developed in this thesis have been presented.
In software requirements area, five proposals have been analyzed, which consider early requirements phase as a source to obtain late requirements [Sant02] [Orti01] [Louc95] [Dijk02] [Bube98]. The main characteristic of these techniques is the analysis and understanding of the business processes before considering the construction of an information system that automates certain business processes. Unfortunately, the majority of these works focuses only on the definition of notations to represent early and late requirements, but only limited research efforts have been made to provide systematic approaches to generate requirements models form business models.
Two proposals were analyzed in the field of conceptual modeling [Alen03] [Orti01], which have the objective of obtaining conceptual

models from the understanding of the organization context. At the present time, there are only few research works focused on providing a methodological solution to the problem of appropriately translate the business model elements into the conceptual schema elements, of the information system. We argue that this is a fundamental activity in the software development process. It is necessary, for the translation between models to be carried out in a methodological process, to assure its application in real software development environments.

The methodological approach presented in this thesis puts emphasis on the use of business models as a starting point of the process to obtain a conceptual schema from the information system and a requirements model, for the software system-to-be. This proposal allows representing the different alternatives to satisfy the business goals, as well as the analysis of the impact that the automation of plans will have on the quality factors expected by the enterprise[1].

This is the reason why a study of the current methods for goal-based requirements analysis was performed, where some of the more relevant proposals in this area are briefly described.

On the other hand, pattern languages have an important role in the research carried out in this thesis. This is because it allows us to guide the delegation of plans to be automated towards a new organizational model, which includes, in an explicit way, the information system as an actor of the organizational model. Therefore, two fundamental processes make up the process to obtain the requirements model and the conceptual model: goal analysis and implementation of automation patterns. The goal analysis has the objective of identifying the plans that need to be automated. This process is based on the determination of the tasks that allow better satisfy the organizational goals.

Finally, heuristics and algorithms are provided to carry out the translation of the organizational model into the conceptual and requirements models.

[1] The quality factors are the aspects of quality that the enterprise wants to enhance with a software system.

Part I

The Early Requirements

Chapter 3

The early requirements phase

This Chapter describes the goal-based requirements elicitation process proposed in this thesis. The objective of this process is to find the best way to develop business tasks in order to achieve organizational goals. The early requirements phase represents our starting point towards the construction of a software system that automates certain organizational processes.
The Chapter also introduces the basic concepts of the Tropos framework that are used in this proposal.

3.1 Introduction

The early requirements analysis [Fuxm01] [Yu97] is one of the most important and difficult phases of the software development process.

In this phase, the requirements engineer attempts to understand the organizational context, the goals and social dependencies of its stakeholders in order to have the appropriate information to develop the information system-to-be. This phase demands critical interactions with the users; a misunderstanding at this point may lead to expensive errors during later development stages. Not surprisingly, several approaches have been devoted to developing languages and analysis techniques for early requirements analysis (e.g., [Dard93] [Yu97] [Anto96] [Gior05] [Lams01]).
Several research works focus on analyzing the early requirements phase as a source for obtaining software requirements [Cast02] [Maid04] [Bres04] [Jaco95a] [Bube95] [Bide02] [Magn00]. The main feature of these techniques is the analysis and understanding of the organizational processes before building an information system. In these approaches, it is important to determine: a) the role of the software system in the organizational context, b) the users of the software-to-be, and c) the impact of the system in the performance of the organizational processes (Figure 3.1).
This knowledge will help to build a software system that works harmoniously with the organizational processes. We considered that it is not possible to develop a software system that provides real value to the enterprise without the understanding of the context where the system will operate. Goals play a very important role in this phase; goals have been recognized as a basic tool in requirements engineering [Lams01]. For this reason, they have been used in the early requirements phase, and to obtain the functional [Anto97] and the non-functional requirements [Chun00] for a software system.

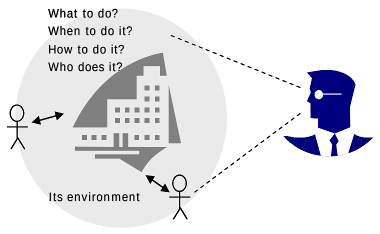


Figure 3.1 The organizational model shows the environment of the business

We ague that, a reason for using goals in the early requirements phase is that they allow the visualization of states that an enterprise expects to achieve. Goals also provide the purpose and reasoning that will justify each one of the requirements of the information system.
The objective of the proposed method is to provide a methodological approach for deriving the software functionality from organizational models. Figure 3.2 shows a general schema of the early requirements phase, which will be explained in this Chapter and Chapter 4. The inputs of our proposal are the goals that the business needs to achieve by implementing a software system. These goals are defined in the actor diagram of the Tropos framework. As result of the method, the software system is included as a organizational actor in the organizational model. In this Chapter, a goal analysis is carried out to determine the set of alternative tasks that better satisfy the business objectives.
One contribution of this thesis is to make the model transformation process systematic by proposing rules to identify the relevant tasks[5] to be automated from the high-level goals of the stakeholders (represented as actors). The proposed approach also allows us to identify the best way to delegate the relevant tasks to the software system actor. The generation process of the late requirements is explained in Chapter 4.

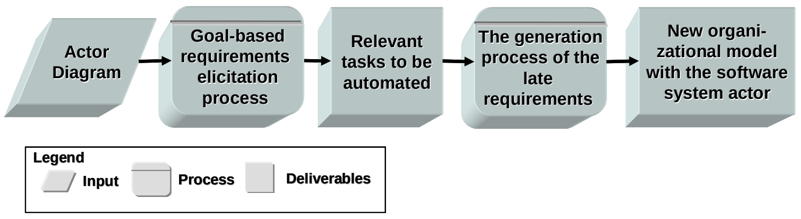

Actor Diagram
Goal-based requirements elicitation process
Relevant tasks to be automated
The generation process of the late requirements
New organi-zational model with the software system actor
Legend
Input
Process
Deliverables

Figure 3.2 The early requirements phase processes

[5] The word “relevant” has been used in this thesis to indicate those elements whose automatic execution satisfies business goals in the most appropriate way.

3.2 The early requirements phase

In [Yu97], a distinction is made between early and late requirements phases. The early requirements emphasize an understanding of the *whys* of the business, while late requirements emphasize *what* the system should do and *how* do it [Yu94]. Thus, the early requirements phase consists of analyzing and identifying the stakeholders and their intentions. Stakeholders are modeled as social actors who depend on each other for goals to be achieved, plans to be performed, and resources to be furnished. Intentions are modeled as goals which are decomposed into finer goals through a goal-oriented analysis. These finer goals can eventually support evaluations of alternatives [Bres04].

In recent years, there is an increasing number of works devoted to obtain requirements specifications from the understanding of a business setting. The reason of this increasing interest is based on the following reasons [Yu97]:

- System development involves many assumptions about the embedding environment and task domain. As discovered in empirical studies (e.g., [Curt88]), a poor understanding of the domain is a primary cause of project failure. To have a deep understanding about a domain, there must be a clear understanding of interest priorities, and abilities of various actors and players, in addition to a good grasp of the domain concepts and facts.
- Users need to help coming up with initial requirements, in the first place. As technical systems increase in diversity and complexity, the number of technical alternatives and organizational configurations constitute a vast range of options. A systematic framework is needed to help developers understand what users want and to help users understand what technical systems can do. Many systems that are technically sound have failed to address real needs (e.g., [Grun99]).
- Software systems are increasingly expected to contribute to the redesigning of organizational processes. Instead of automating well-established organizational processes, systems are now

viewed as “enablers” for innovative organizational solutions (e.g., [Hamm93]). More than ever before, requirements engineers need to relate systems to business and organizational objectives.

- Having well-organized bodies of organizational and strategic knowledge would allow such knowledge to be shared across domains at this high level, deepening the understanding about relationships among domains. This would also facilitate the sharing and reuse of software (and other types of knowledge) across these domains.
- As more systems in organizations interconnect and interoperate, it is increasingly important to understand how systems cooperate (with each other and with human agents), to contribute to organizational goals. The early requirements models that deal with organizational goals and stakeholder interests cut across multiple systems and can provide a view of the cooperation among systems within an organizational context.

Now that the importance of the early requirements has been explained, below, we present the foundations of the early requirements phase in detail.

3.3 The foundations of the early requirements phase

This section presents the main concepts that are used in the early requirements phase: Goal modeling and organizational modeling. Both approaches have been combined to create a goal-based requirements elicitation process. The differences of our proposal from other proposals are presented in this section.

3.3.1 Goal modeling

The need to model *why* a system should be developed has been recognized since the early days of requirements engineering [Ross77]. However, most requirements modeling notations and techniques focus only on the late phase of the requirements engineering process.

Methods supporting analysis of this kind include semi-formal methods (e.g. structured methods [Ross77], object-oriented methods [Rumb91], [Rumb98b] [Past01]) and formal methods (e.g. model checking [Alpu05] [Clar96]), there are also other methods that focus on scenarios [Leit97] [Some05] and aspects [Arau03] [Samp05], [Grun99], etc.

Goal modeling is intended to address the early-phase of requirements engineering, in which stakeholders and goals are explored and alternative system proposals that satisfy the goals are investigated [Lete04].

Nowadays, several research efforts use goal mechanisms during the requirements elicitation process. One of the most relevant works in this field is the KAOS approach [Lete04] [Lams01] [Dard03]. KAOS provides formal rules for deriving requirements from goal descriptions. It is based on theories of formal specification languages to analyze functional and non-functional requirements. However, the use of this approach is restricted to analysts that are used to deal with formal methods as a current concept in their modeling activities. KAOS also provides support for finding alternatives to satisfy the organizational goals. Another goal-oriented method is GBRAM (Goal-Based Requirements Analysis Method) [Pott94] [Anto97], which is focused on the generation of operational requirements from high-level goals. However, this method does not establish a clear distinction between the information used in the early and late requirements phase [Yu97].

As a consequence of this problem, GBRAM does not have a clear representation of the complete process for software development. Another important research work on goal modeling is the NFR Framework proposed in [Chun00]. This approach focuses on analyzing the impact of non-functional requirements in the software development process.

The main difference among the current goal-based approaches and the proposed approach is the use of a systematic method that guides the analyst in the construction of an information system. This proposed method puts emphasis on the early and late requirements phases.

Some goal concepts found in the literature and some advantages of using goal modeling are the following:

- Goals are objectives that the system must achieve. The word "*system*", here, refers to the software-to-be together with its environments [Fick92] [Zave97].
- Goals are targets that provide a framework for the desired system [Anto96].
- A goal is a desired property of the environment [Robi04].

In many goal-oriented works, the importance of the goals in software development is emphasized [Anto97] [Dard03]. Some of the advantages are the following

- Goals make the relationships between the operations of the business and the high-level goals outlined by the administrators explicit.
- Goals provide a precise judgment to determine the relevancy of requirements. A requirement is pertinent regarding a group of goals, if its specification is used to satisfy at least to one of the goals.
- Goals can be used to determine the organizational process necessary to satisfy each goal.
- Goal refinement provides a natural mechanism to structure complex requirements documents and to increase their legibility.
- Goals can be used as a complete and effective way to determine the specification of requirements. The specification is complete regarding a group of goals if all the goals in the group can be satisfied with determined requirements.
- Goals can be used to identify and solve conflicts among different points of view about the way of satisfying a goal.

3.3.2 Organizational modeling

Organizational modeling is a set of techniques used to represent and structure the knowledge of an enterprise [Bube94]. Organizational analysis allows us to precisely determine the following aspects: The operations that satisfy each one of the goals, the network of dependencies among actors, the sequence in which the tasks of each organizational process should be executed, the dependency type, the tasks to be automated, etc. This information is fundamental for the generation of a requirements model that gives real support to organizational tasks.

There is a lot of research being done in this field [Yu95] [Bube95] [Cesa02], [Louc95] [Cast02]. We have chosen the Tropos methodology to represent the organizational environment because it supports the early requirements and allows us to analyze the processes that involve multiple participants (both humans and software systems) and the intentions that these processes are supposed to fulfill. The methodology is defined in terms of the concepts of *agent*, *goal*, and related abstract notions. These notions are used to support all software development phases, from early requirements analysis to implementation. The following sub-section describes this framework in detail.

3.3.3 Tropos Framework

This thesis is conducted within the context of the Tropos methodology, which adopts the concepts of the *i** modeling framework [Yu95], whose aim is to construct and validate a software development methodology for agent-based software systems. One of the main advantages of this methodology is that it allows us to capture not only *what* or *how*, but also *why* a piece of software is developed. Tropos, in return, provides a more refined analysis of the system dependencies and a well defined mechanism to deal with functional and non-functional requirements.

The main difference in the early requirements analysis carried out in the Tropos framework and our proposal is that the Tropos methodology focuses on the representation of the future state of the business, starting with the high-level goals of business, and determining the group of alternatives to fulfill these goals. Our proposed methodology focuses on eliciting the current state of an existing business to determine the organizational plans, whose automatic performance would best satisfy the goals of the business. These plans can be considered as requirements of the system-to-be.

We have used the Tropos notation to represent and analyze the early requirements. This framework uses the following graphical representations to represent the organizational environment [Sann02]

- **Actor Diagrams** This is a graphical representation where actors and their goals, and the dependencies among actors, are shown. This model emphasizes the static aspects of the enterprise.
- **Goal Diagrams** This is a graphical representation where the goals, plans, and dependencies of each actor are analyzed in depth.

In the following paragraphs, the key concepts and the diagrams used in our proposal of the Tropos framework are presented. In [Sangt02] and [Bres04] the Tropos Framework is presented in detail.

Concepts and Notation

Tropos adopts Eric Yu's *i** model [Yu95] which offers *actors* (*agents*, *roles*, or *positions*), *goals*, and *actor dependencies* as primitive concepts. The five basic concepts in the Tropos Framework are the following [Sant02]:

Actor. An actor is an entity that has strategic goals and intentionality within the system or the organizational setting. An actor represents a physical or a software agent, as well as a role or position. A goal graph can be associated to an actor by circling the graph with a dashed line.

Hardgoal/Softgoal. This represents actor strategic interests. *Hardgoals* are distinguished from *softgoals*. There, the second having no clear-cut definition and/or criteria for deciding whether they are satisfied or not. The *hardgoals* are illustrated as a rounded-cornered rectangle, while *softgoal* are illustrated as a cloud.

Plan. It represents a way of doing something at an abstract level. The execution of plan can be a mean to satisfy a goal or a *softgoal* (illustrated as a hexagon).

Resource. It represents a physical or an informational entity (illustrated as a rectangle).

Figure 3.3 illustrates the graphical notation for these modeling concepts.

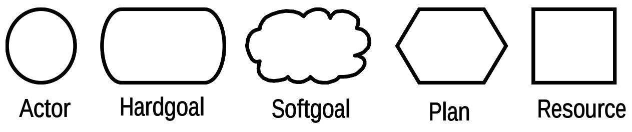


Figure 3.3 Graphic notations of the basic concepts

Dependency. A relationship between two actors, which indicates that one actor, depends for some reason, on the other actor, in order to attain some goal, execute some plan, or deliver a resource. The former is called the *depender*, while the latter is called the *dependee*. The object around which the dependency centers is called *dependum*. In general, by depending on another actor for a *dependum*, an actor is able to achieve goals that it would otherwise be unable to achieve on its own, or not easily, or not as well. At the same time, the *depender* becomes vulnerable. If the *dependee* fails to deliver the *dependum*, the *depender* would be adversely affected in its ability to achieve its goals.

Goal dependency. It is a relationship in which an actor depends on another actor to fulfill a goal, without prescribing the way in which it should be carried out.

Resource dependency. It is a relationship in which an actor depends on another actor to deliver a resource that can be either material or informational.

Plan dependency. It is a relationship in which exist a dependency to carry out of a task, establishing the way in which it should be performed.

Softgoal dependency. This is similar to the goal dependency, with the difference that the goal can not bee precisely defined.

Contribution. It is a relationship between goals or plans representing how goals or plans can contribute (positively or negatively), in the fulfillment of the goal.

The graphical representation of the Tropos dependencies is illustrated in Figure 3.4.

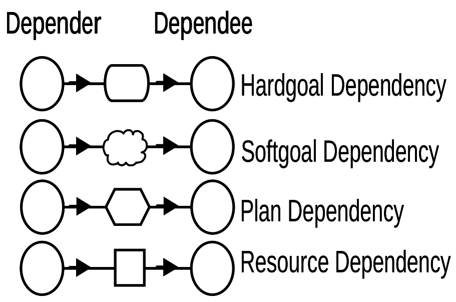


Figure 3.4 Graphic notations of the dependency relationships

Decomposition. It is a relationship between goals or plans representing AND/OR decomposition of root goal/plan into sub-goals/*subplans*.

Means-end. It is a link to join plans with goals. Different alternatives are allowed as means of the relationship. Figure 3.5 shows the intentional relations.

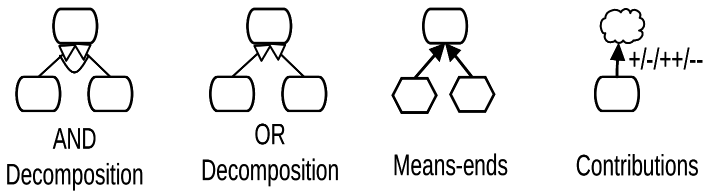


Figure 3.5 Graphic notations of the intentional relations

Actor Diagram

The main objective of this diagram is to have a static view of the environment and the system to be developed. This diagram is made up of the organizational actors, who are associated to other actors by dependency relationships. The actor diagram can also extend the basic concepts of the actor through the refinement of the notions of Role, Position and Agent [Yu00], where:

A role is an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. Dependencies are associated to a role when these dependencies apply, regardless of who plays the role.

An agent is an actor with specific physical manifestations, such as a human. An agent has dependencies that apply regardless of what role he/she/it happens to be playing. We use the term *"agent"* instead of *"person"* for generality, so it can be used to refer to human as well as artificial (hardware, software, or organizational) agents.
A position is intermediate in abstraction between a role and an agent. It is a set of roles typically played by one agent. Positions can *cover* roles, agents can *occupy* positions, and agents can also *play* roles directly.
Association is a set of roles, positions and agents interconnected by *"plays"*, *"Occupies"* and *"covers"* relationships.
The *"INS"* construct represents the instance-and-class relation. The *"ISA"* construct expresses conceptual generalization/specialization. These constructs are used to simplify the presentation of strategic models with roles, positions, and agents. Roles, positions, and agents can be decomposed into sub-parts.

Goal Diagrams

The goal diagram provides a microscopic view of the application domain. Its purpose is to determine some strategies to fulfill the actor's goals, using three basic reasoning techniques *Means-end* analysis, *contribution* analysis, and *AND/OR* decomposition. Specifically, *means-end* analysis helps in identifying plans, resources and *softgoals*, that provide means for achieving a goal. Contribution analysis identifies goals that can contribute positively or negatively to the fulfillment of the goal to be analyzed [Bres04].

3.4 Goal-based requirements elicitation process

This section describes our goal-based requirements elicitation process. It first gives a brief overview of the method, then describes the various steps of the method and illustrates its application on a case study, the *Car Rental*.
This case study is a real project of the Care Technology Company, which concerns the organizational modeling for a car rental

enterprise in Spain. Chapter 8 contains the complete description for this case study.

One of the key issues in solving the problem of generating a software system that fulfills user needs is to provide the analyst with mechanisms to represent the goals that represent the states that an enterprise wants to fulfill. In this context, goal modeling plays a relevant role in the software development process, because it permits the determination of different alternatives that exist to better satisfy the goals that fulfill the organizational goals, using a software system. This process provides a deep understanding of the organizational tasks and the reasons *why* tasks are executed.

The importance of the goals becomes evident in requirements engineering; they provide the motivations and reasoning to justify each one of the requirements of the information system. However, in spite of all the advantages that the goals and the multiple works carried out in this area provide, there are many factors that need to be improved to assure their practical application. This Chapter presents our goal-oriented proposal for the elicitation of software requirements to provide an answer to some problems of the current goals modeling approaches.

Overview

The goal-based requirements elicitation process consists of deriving the requirements for a future software system from an organizational context (Figure 3.6). Therefore, the process starts with the definition of an organizational model that reflects the current enterprise situation.

This model must represent the high-level goals and the relevant actors in the business. A goal analysis phase is carried out to identify the relevant tasks that fulfill the goals of the enterprise.

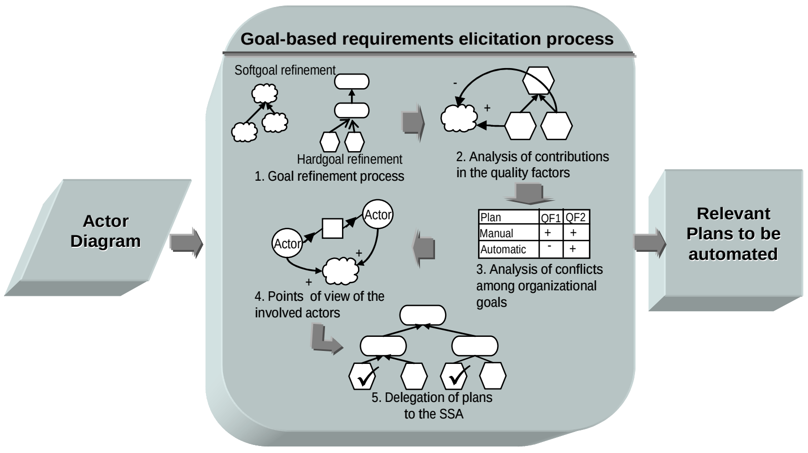

Goal-based requirements elicitation process
Actor Diagram
Softgoal refinement
Hardgoal refinement
1. Goal refinement process
2. Analysis of contributions in the quality factors
Plan | QF1 | QF2
Manual | + | +
Automatic | - | +
3. Analysis of conflicts among organizational goals
Actor
Actor
4. Points of view of the involved actors
5. Delegation of plans to the SSA
Relevant Plans to be automated

Figure 3.6 Goal analysis schema

The proposed analysis is composed of five steps that help to choose the appropriate tasks to-be automated. In the last step (*Delegation of plans to the software system actor*), a pattern language is proposed in order to build an organizational model which includes the software system to-be. At this point, in the late requirements phase, the system is described within its operational environments, its functions, and relevant characteristics. This model is a final result of our proposed method.

In summary, the steps of the goal-based requirements elicitation process to identify the plans that must be automated are the following

- Goal refinement process. The first step consists in carrying out a goal refinement for each goal of the actors. Thus, this step is divided into two sub-processes: *hardgoal* and *softgoal* refinement. The quality factors are also identified by the *softgoals* of the enterprise.
- Analysis of contribution in the quality factors. The second step consists of analyzing the plans and goals that best satisfy

the quality factors. Therefore, each atomic plan[1] found in the organizational model must be divided into two *subplans* (execution by a software system or manual execution). The impact of these *subplans* with the quality factors is analyzed in order to determine the plan that better contributes to the satisfaction of the organizational goals.

- Analysis of contradictions among organizational goals. The third step consists of identifying the contradictions among the organizational goals, once the contribution analysis has been carried out.
- Point of view of the involved actors. The fourth step consists in resolving the contradictions among the goals. Therefore, an analysis of the point of view of the actors involved in the achievement of the goals is carried out. The relevant plans to be automated must be identified.
- Delegation of plans to the Software System Actor (SSA). The last step consists of delegating the relevant plans to the Software System Actor. This step is explained in depth in the next Chapter.

Finally, the goal-based requirements elicitation process is further discussed in the next sub-sections.

3.4.1 Goal refinement process

The use of goal analysis mechanisms in software requirements has been discussed in the literature by several authors [Dard03] [Anto97] [Pott94] [Chun00] [Lete04] [Robi04].

In this modeling context, the Tropos methodology provides one of the most well-established founded techniques for goal analysis [Gior05]. Tropos provides not only informal notations for representing goals, but it also provides a well-established framework to permit formal reasoning about the goal models.

[1] Atomic plans are those plans that do not need to be divided into other subplans to be executed.

The first phase of the elicitation process is the goal refinement process. The objective of this goal refinement is to decompose each high-level goal of the enterprise into more specific sub-components, until the desired level of specific plans for satisfying the goal is reached.

The steps to carry out the goal refinement in the proposed method are detailed below, and the *Car Rental* case study is analyzed, in order to illustrate the goal refinement process.

Step 1. Build an actor diagram that shows only the general goals and dependencies among the actors. A general goal reflects the state of affairs that an actor wants to fulfill.

Step 2. Each general goal of the actor diagram must be refined (in the boundary of the analyzed actor) using *AND/OR* decomposition, *means-end*, or contribution links in order to determine the low-level goals that satisfy the objectives of the enterprise.

The goal refinement process ends when the current plans (represented as hexagons) performed by the organizational actors are linked with the sub-goals identified in the goal-refinement process.

The links that are used to make the refinement goals are detailed below:

The AND *decomposition* links are used to represent the set of sub-goals ($G_{1.1}$, $G_{1.2}$ …) that satisfy goal G_1.

The OR *decomposition* links are used to represent alternatives that satisfy goal G_1.

The *means-end* links are used to represent the different *means* that exist to fulfill an *end* (usually a goal). This link allows us to relate a goal to a set of plans ($P_{1.1}$, $P_{1.2,}$…) that represent the alternatives that satisfy the goal; however the means can also be sub-goals ($G_{1.1}$, $G_{1.2,}$…).

The contribution links are used to specify the positive or negative contributions of a goal, or plan, to a *softgoal.*

Figure 3.7 shows the links used to perform the goal-refinement process.

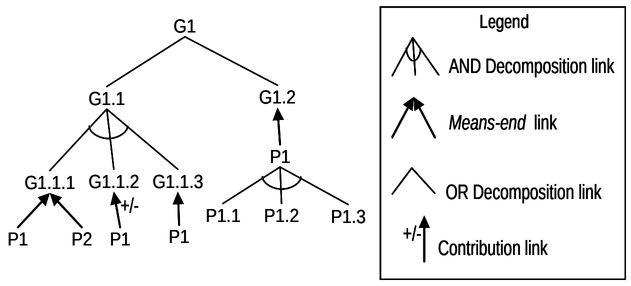


Figure 3.7 Refinement links

Step 3. Each *softgoal* of the actor diagram must be refined. This refinement is carried out in the same way as the *hardgoal* refinement. However, the refinement of this goal type does not conclude with the determination of plans that fulfill the goal, but rather the *softgoal* refinement process concludes with the determination of means that satisfy the *softgoal*.

The *softgoals* are used to represent the quality factors that the enterprise wants to fulfill. Quality factors will help the organization to improve the performance of organizational processes and management systems. In the literature there are many quality attribute taxonomies [Boeh96] [Boeh78] [ISO01].

A set of quality factors is detailed below. It is important to point out that the quality factors analyzed in this modeling phase are directly concerned with measures for the organizational processes, rather than measures for the information system to-be.

- ***Competitiveness*** This quality factor refers to the characteristics (profitability, costs, and quality) that permit an enterprise to compete effectively with other firms.
- ***Performance*** This quality factor refers to the response and processing times of the organizational processes.
- ***Security*** This quality factor refers to the ability to prevent unauthorized access to the information used by the enterprise.

Once the refinement of the *general* goals has been carried out and the quality factors required by the company have been identified, the next step is the contribution analysis between the elicited plans and the selected quality factors.

Example

The evaluation of our methodological approach has been done with several case studies. *The Car Rental* case study is used in this Chapter in order to illustrate our proposal.

The first step of the Goal-based requirements elicitation process is related to the construction of the organizational models, this is carried out in the goal-refinement process. Figure 3.8 shows a partial view of the actor diagram. In this model, the *general* goals (represented as ovals) of the actors (*Customer*, Company employee, Associated branches, Mechanic, Insurance) are shown. Actors who play a role in the enterprise are also depicted, e.g. the *Customer* actor, who wants to rent or buy a car.

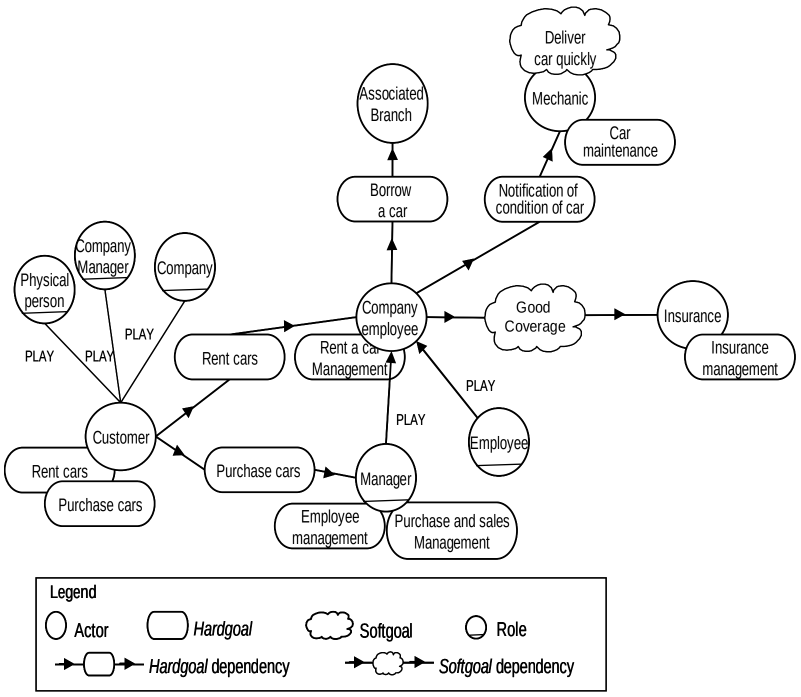


Figure 3.8 Partial view of the actor diagram for the *Car Rental* case study

This actor plays several roles in the organizational model: a) Company, who has an agreement for a lower price*;* b) Company Manager, who works for the company, or c) Person, who is different from the last two roles. The arrows indicate the dependency relationships between actors. For example, the *Customer* depends on the *Company employee* to rent a car.

The construction of the goal diagram is carried out by refining each general goal. Figure 3.9 presents a partial view of the goal diagram for the *employee* actor. In this example, the *general* goal of the Employee actor *cars reservation management* is refined into alternative sub-goals: 1) C*arry out reservations directly in the branch*, and 2) C*arry out reservations using alternative ways* (such as internet or phone reservations).

The goal *Carry out reservations directly in the branch* is refined into three sub-goals using a*nd* decomposition *Analyze Customer, Analyze the car availability***,** and *Formalize reservation*. The goal-refinement process ends when the plans (represented as hexagons) for fulfilling the goals are identified. Once the refinement of the *general* goals has been carried out and the quality factors desired by the company have been identified and decomposed, the next step in the proposed method is focused on the analysis of contributions between the elicited plans and the selected quality factors.

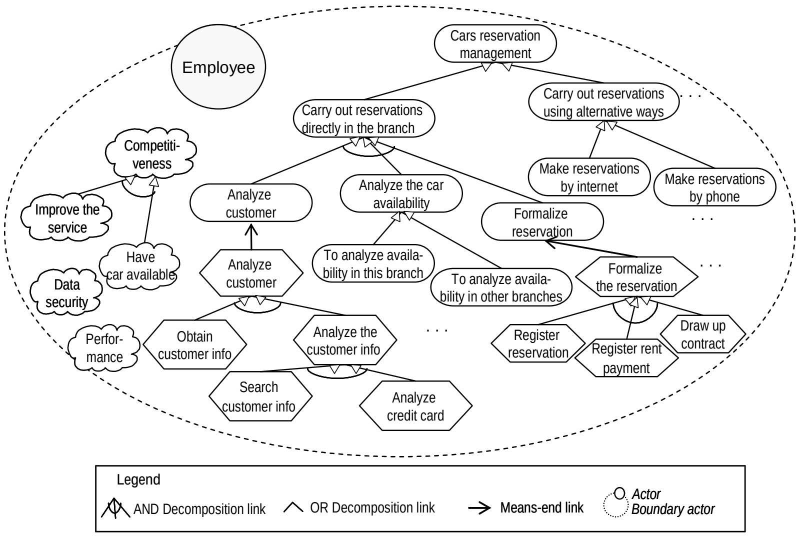


Figure 3.9 Partial view of the actor diagram for *The Car Rental* case study

3.4.2 Analysis of contributions in the quality factors

The second step of the goal-based requirements elicitation process is the analysis of contributions in the quality factors. Contributions describe the (positive or negative) influence of a goal or task on the satisfaction of a quality factor (*softgoal*).

The *softgoal* contribution analysis is one of the key factors in the goal analysis process because it allows us to identify the plans and goals that better satisfy the quality factors.

The contributions describe the influence of a goal or plan on the satisfaction of a *softgoal*. The values of the contributions are positive contribution (+), negative contribution (-), full satisfaction (++), fully denied (--) [Gior05].

The analysis of contribution in the quality factors is carried out by a set of steps, which are detailed as follows:

Step 1. Propagation of the *Atomic* plans. The first step of this process consists of propagating each *atomic plan* in the goal diagram into two alternative *subplans* manual execution or automatic execution. The automatic execution of a plan represents the execution through a software system that automatically performs the organizational plans.

Step 2. Associating the plans with the quality factors. The second step consists of associating the plans with the selected quality factors. Therefore, we must determine the positive or negative contribution of manual and automatic execution of the plan with the quality attributes that the business wants to fulfill.

Step 3. Contribution analysis. Finally, the third step consists of identifying the plans that best fulfill the quality factors. In this phase the contradictions and conflicts between the plans and goals must be identified.

Example:

In the *Car Rental* case study, Figure 3.10 presents a fragment of the model with the quality factors contributions for the *Employee* actor. The plans *Search Customer info* and *Analyze credit card* have been propagated in two *subplans* in order to represent the manual and automatic execution of these plans.

For each propagated plan, the contribution links are created to associate the plans with the quality factors. This is done in order to identify the influence of the plans with the quality attributes. For example, the manual execution of the plan *"Analyze credit card"* has a negative contribution on the Performance attribute. We consider that the selection of the correct plan to be automated is not always a trivial task. This is because the contribution analysis gives rise to contradictions among the alternatives to satisfy the quality factors.

The conflicts analysis is explained in the following section.

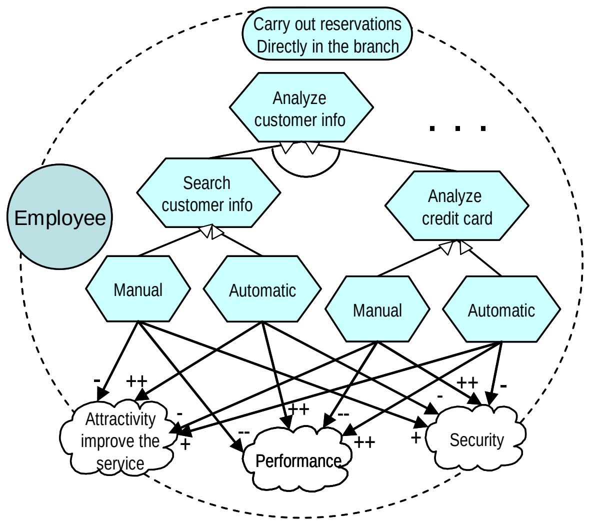


Figure 3.10 Quality factor contributions

3.4.3 Analysis of conflicts among organizational goals

The third step of the elicitation process is the analysis of conflicts among organizational goals. This analysis is carried out after associating the plans and the quality factors by the contribution link.

As mentioned above, the selection of the plans to be automated is not a trivial task. Sometimes, the enterprise employees do not have a clear idea of the best way to satisfy the organizational goals. This is because, in most of the cases, the employees do not have a global view of the enterprise and the goals that the enterprise wants to fulfill.

The analysis of contributions is useful for representing a global view of the organization, which allows us to evaluate, the objectives of the business and how they are achieved.

More specifically, this model is useful for analyzing the different alternatives for fulfilling the organizational goals through the automation of the organizational tasks using a software system.

The steps to perform the contradictions analysis are presented below:

Step 1 Create a matrix with the *Atomic* plans and the quality factors. The plans placed in the matrix are relevant plans, which the analysts want to analyze.
Table 3.1 shows the matrix of contributions used for analyzing the conflicts among organizational goals. Columns represent the quality factors that the business wants to achieve. Rows represent the plans to-be-analyzed. Each plan is analyzed in two options, the first is the execution in a manual way, and the second option is the automatic execution of the plan.
Step 2 Place the value of the contributions in the matrix. The values (++, --, +, -) of the contributions are those identified in the contributions links.
Step 3 Compare the contributions to of the quality factor for each propagated plan.

Table 3.1 Matrix of contributions

Plans	Quality factor 1	Quality factor 2	Quality factor n
Plan 1 executed manually			
Plan 1 executed automatically			
. . .			
Plan n executed manually			
Plan n executed automatically			

Example:
In order to resolve the contradictory cases, the kind of contributions of the propagated plans (manual or automatic options) with the selected quality factors.
Following with the running example, Table 3.1 shows the matrix that includes the plans *Search Customer info (*plan 1) and *Analyze credit card* (plan 2). These plans have been analyzed considering the two alternatives, automatic and manual execution. When the

contributions plans were analyzed, some contradictions between the quality factors and plans were detected *(*Figure 3.10*)*.

The plan *Search Customer info (automatic)* positively contributes to the quality factor *Performance*; however, this plan has a negative contribution to the quality factor *Security*. In this last case, the best option to select is the manual execution of the plan. In order to solve this conflict, the analyst must determine the priority quality factors for choosing the plans that need to be automated. Another possible solution consists of taking into account the point of view of the actors involved in the plans. This solution is analyzed in the following step.

Table 3.2 Example of the matrix of contributions

Plans	Competiti- veness	Performance	Security
Search *Customer* info (Manual)	-	--	+
Search *Customer* info (Automatic)	++	++	-
Analyze credit card (Manual)	-	--	++
Analyze credit card (Automatic)	+	++	-

3.4.4 Points of view of the involved actors

The fourth phase of the elicitation process is the analysis of the point of view of the actor involved. The satisfaction of the goals of specific actors can be affected not only by the execution of their own plans, but also by the execution of the plans of other actors.

The Tropos Framework uses the concept of dependency to represent social and intentional relationships among the actors. A dependency is a link between two actors, where an actor, for some reason, depends on another actor to attain goals, execute plans, or deliver a resource. The former actor is called the *depender*, while the latter is called the *dependee*. The object around which the dependency relationship centers is called the *dependum* [Yu95].

The dependency relationships allow us to represent the collaboration among organizational actors. In this proposal, the dependencies are analyzed to provide useful information to the analysts so that they can make decisions about the plans that must be automated.

In order to take these decisions, the contributions of the actor's plans to the quality factors must be taken into account, as well as the contributions of the actors associated through dependency relationships.

As a result of the previous steps (goal refinement process, analysis of contributions in the quality factors, analysis of conflicts among organizational goals and points of view of the involved actors), the plans that better satisfy the quality attributes of the enterprise have been identified. These relevant plans represent the requirements to be considered in the construction of the software system.

Next, an example of the points of view of the involved actors step is presented, where a partial view of the organizational model with the selected relevant plans to-be-automated is shown.

Example

Figure 3.11 presents an example of the analysis of the points of view of the involved actors for the *Car Rental* case study. The objective of the process shown in this example was to determine the best way to carry out the payment of a rented car. In this example, the *employee* actor is related to the *Customer* actor by the *payment* resource dependency. Therefore, we analyze the contributions of the actor *Customer* with a specific quality factor (*security*). In this case, it is possible to determine that in both cases (*employee* actor and *Customer* actor) the execution of the *payment* plan in an automatic way contributes negatively to the *security* factor. Therefore, in this specific case, the best option to be selected is the manual execution of the plan *Register rent payment*. Following with the example of the case study, Table 3.3 shows the matrix of contributions of the *Employee* actor detailed in the actor diagram of Figure 3.9.

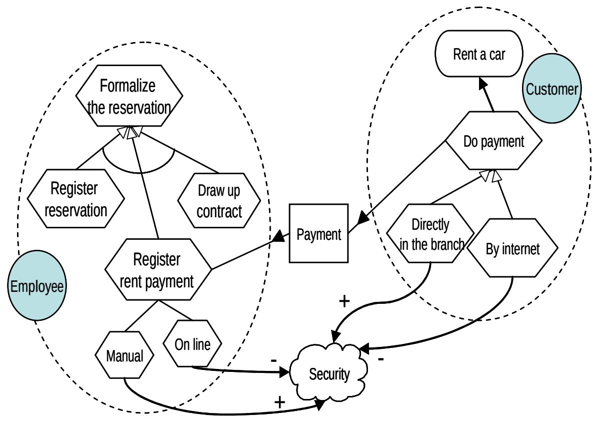


Figure 3.11 Analyzing points of view of the involved actors

This table contains all the plans of the model, which are analyzed in two alternative solutions: automatic and manual execution. Once the relevant plans have been represented in the matrix, then we must analyze each plan to determine its positive or negative contribution to the quality factors desired by the enterprise.
The matrix that associates relevant plans and contributions with quality factors is obtained as a final result of the previous steps of this proposal (analysis of contributions in the quality factors, analysis of conflicts among organizational goals and analysis of the point of view of the involved actors).
Using the generated matrix, the plans to be automated can be determined by comparing the positive contributions of both solutions, manual and automatic task execution. Table 3.3, shows the relation among the alternatives to satisfy the organizational plans and the proposed quality attributes. The table also shows the plans selected to be automated according to their positive contribution to the quality factors.
As a result of the previous steps (goal refinement and contribution analysis) the plans that best satisfy the quality attributes of the enterprise have been identified. These relevant plans represent the requirements to be considered in the construction of the software

system. Figure 3.12 shows a fragment of the organizational model where the relevant plans to be automated have been remarked.

Table 3.3·Contributions matrix for *The Car Rental* case study

Plans	Competitive ness	Perfor mance	Security	
Obtain C*ustomer* info (Manual)	-	-	-	
Obtain *Customer* info (Automatic)	+	+	+	*To be automated
Search *Customer* info (Manual)	-	--	+	
Search *Customer* info (Automatic)	++	++	-	*To be automated
Analyze credit card (Manual)	+	--	++	
Analyze credit card (Automatic)	+	++	-	
Obtain reservation info (Manual)	-	-	+	
Obtain reservation info (Automatic)	++	++	+	*To be automated
Search car availability (Manual)	-	-	+	
Search car availability (Automatic)	++	++	+	*To be automated
Register reservation (Manual)	-	-	+	
Register reservation (Automatic)	++	++	+	*To be automated
Register rent payment (Manual)	-	-	+	
Register rent payment (Automatic)	+	+	-	*To be automated
Draw up contract (Manual)	+	+	+	
Draw up contract (Automatic)	+	+	-	
Request car garage (personality)	-	-	+	
Request car garage (by phone)	+	++	+	

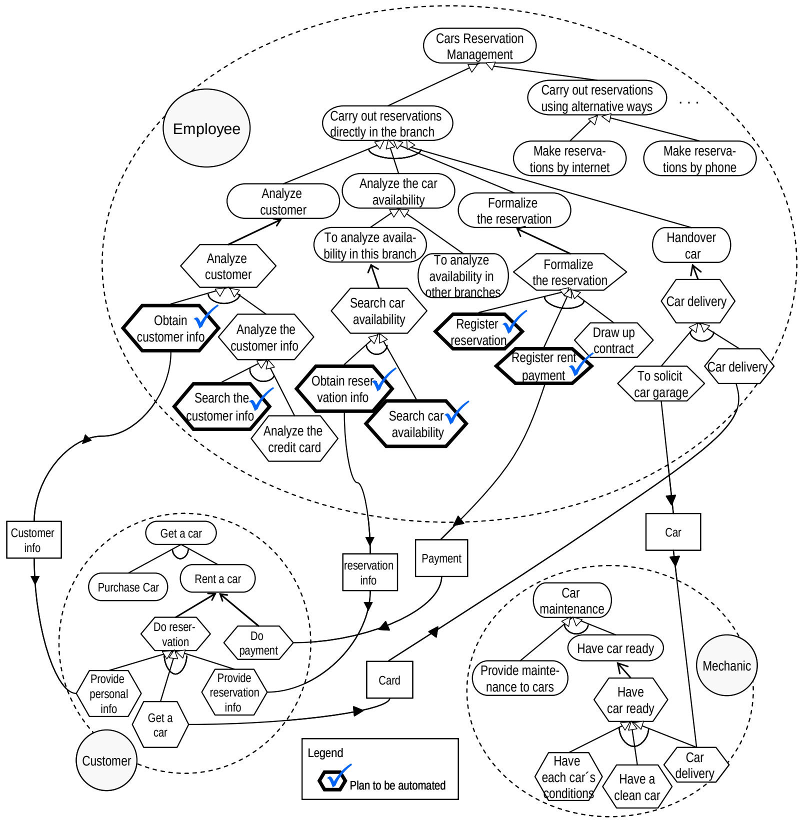


Figure 3.12 Partial view of the organizational model with the selected relevant plans

3.4.5 Delegation of plans to the software system actor

The last step in the goal-based requirements elicitation process is the delegation of plans to the software system actor. One of the key capabilities of the Tropos framework is the inclusion of the software system within the organizational context. To do this, the software system actor is placed as an organizational actor in the goal diagram of the business. At this point, the plans that better satisfy the quality attributes of the enterprise have been identified. These relevant plans represent the requirements to be considered in the construction of the

software system. Therefore, the objective of the phase is to delegate all the relevant plans to the software system actor.
Afterwards, the plans and resources needed to accomplish the goals are then redirected towards the system actor. Therefore, the satisfaction of the goals will not be altered; only the actor responsible for its fulfillment is modified. The internal plans in the software system actor must be defined in order to satisfy its goals.
As a result of this process, a new organizational model that represents the relationships among the software system actor and the organizational actor is generated. The definition of this new model is carried out in a systematic way through a pattern language, which is explained in the next Chapter.

3.5 Summary

This Chapter defines a goal-based requirements elicitation process that allows us to identify the relevant plans to be automated. To do this, the high-level goals (that fulfill the objectives of the business) are refined until the level of specific plans for satisfying the goals is reached. In this process, the following elements are analyzed: a) the contributions in the quality factors, b) the conflicts among organizational goals, and c) the point of views of the involved actors.
The next Chapter describes the analysis carried out in the early requirements phase, where a pattern language is proposed to delegate the relevant plans, towards a new organizational actor that represent the software system to-be (this has been explained briefly in section 3.4.5).

Chapter 4

Joining early and late requirements

This Chapter describes a method to reduce the abstraction level between the early requirements and late requirements by creating a new intermediate organizational model that contains the relevant information to be automated by the software-to-be. This process is guided by a pattern language called FELRE (*From Early Requirements to Late Requirements*).

4.1 Introduction

This Chapter presents the last step of the goal-based requirements elicitation process proposed in this thesis. In this step, the relevant organizational plans to be automated, which were identified in previous goal analysis, are delegated towards a new actor that represents the software system. The delegation is carried out by a set of patterns that analyze the several possibilities that exist for delegating relevant information to the system actor.

A pattern reflects something that has been used in a number of situations and, thus, has some generality. The description of a pattern contains a context, which explains the intent of the pattern and suggests how it must be used. Patterns also express solutions in ways that allow some variation, depending on the details of a circumstance. Finally, pattern descriptions can express architectural considerations, independently of specific languages and design methodologies.

We have used a set of design patterns in order to transform an organizational model (which represents the stakeholders and their associated goal) into the functional and non-functional requirements for the system-to-be. The process includes heuristics for identifying relevant tasks to be automated from stake holder’s goals, and also to identify the best way to delegate the relevant tasks to the system-to-be. In order to make the process systematic, a set of patterns is defined which specifies the possibilities that exist to delegate organizational plans towards the software system actor. Then, the system-to-be and its components are represented as a system actor, who will be the responsible actor for fulfilling the assigned relevant tasks. All the transformational steps proposed in this thesis were implemented using a model-driven approach. This enables us to reduce the abstraction level of a “pure” organizational model so that it is closer to the software requirements model. The proposed method complies with the MDA approach, implementing the concept of PIM (platform independent model)-to-PIM transformations.

It is important to point out that we have only used a pattern-based approach in the phase of the delegation of plans to the software system actor. This is because we consider that the steps of the transformational process can be systematically defined; this enables us to define a set of recurrent solutions for each of the steps of the transformational process.
Rules and algorithms to guide the transformational process have been used to perform other transformations between models where not systematic steps were detected.
The structure of this Chapter is as follows, in the second section, a brief description of the model driven architecture is presented; next section shows an introduction of the proposed patterns; and third section presents the concepts used in the proposed pattern language, also, the set of patterns and the pattern language are outlined in this section. Finally, the summary of the Chapter is presented in last section.

4.2 The model driven architecture

In recent years, Model Driven Architecture (MDA^{TM})[1] [OMG01] has been proposed to support the development of large software systems providing an architecture where systems can evolve, and technologies can be integrated and harmonized.
MDA is an approach to system development, which increases the power of models in that job. It is *model-driven* because it provides a mean for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.
The MDA separates certain key models of a system, and brings a consistent structure to these models. Platform Independent Models (PIMs) which can be transformed into one or more Platform Specific *Models* (PSMs). This allows the system to be implemented in different platforms, while still maintaining the same PIM.

[1] MDATM is a trademark of the Object Management Group.

The term Platform-Independent Model is used to refer to a model that has only the structure and functionality of a system and no information about implementation details. Platform-Specific Model is used to refer to models that have information about implementation details [OMG01]

The MDA Guide Version 1.0.1 describes several transformation methods. Here, we limit to describe the model transformation applied at this stage of the thesis. Figure 4.1 illustrates this type of transformation, where, an organizational model specified in the Tropos Framework is created, that represents the initial PIM (Organizational Model "*Pure*") of the proposed method. This model will use a platform independent modeling language. Then, a pattern language is used in order to transform the original organizational model into other organizational model which includes the software system actor. This model will also have a platform independent modeling language and it will be the new PIM (New Organizational Models with the Software System Actor) obtained.

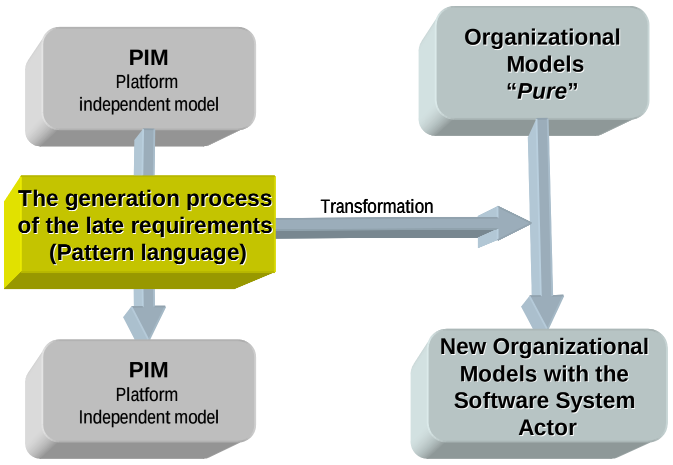

PIM
Platform independent model
Organizational Models "Pure"
The generation process of the late requirements (Pattern language)
Transformation
PIM
Platform Independent model
New Organizational Models with the Software System Actor

Figure 4.1 MDA schema

4.3 Pattern languages

Patterns are a well-known and broadly used technique to specify software design and implementation. Analysis patterns usage is rapidly growing in the software engineering community. This approach has recently been applied in the area of information system engineering, particularly by those advocating object-oriented development approaches and reuse. They are also used in: software programming, software design, data modeling, and systems analysis.
Most of the existing work on patterns has been influenced by the book of Christopher Alexander "The Timeless Way of Building" [Alex79]. This book describes the importance of patterns in such a way that the basic principles of patterns are applicable to other fields as well. According to Alexander, patterns are *"a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"*. Patterns-based approaches have been established in software programming, software design, data modeling, and in systems analysis. Therefore similar definitions of the term "*pattern*" found in the literature are:
"A pattern is a description of a common solution to a recurrent problem, which can be applied to a specific context" [Gamm95].
"… An idea that has been useful in one practical context and will probably be useful in others" [Fowl97]
"… The static and dynamic structures of solutions that occur repeatedly when producing applications in a particular context" [Copl95].
In the area of business development, patterns are relatively new and untested. In the context of organizational development, Coplien [Copl95] argues that *"patterns should help us not only to understand existing organizations but also to build new ones"*. However, patterns rarely stand alone. Each pattern works within a context, and transforms the system in that context to produce a new system in a new context.

Therefore a collection of patterns falls short of being a pattern language.

On the other hand, a pattern language can be detailed as: *a cascade or hierarchy of parts, linked together by patterns that solve generic recurring problems associated with the parts*. Each pattern has a title and, collectively, the titles form a language for design[Copl95].

"… A *pattern language* defines a collection of patterns and the rules to combine them into an architectural style.

Pattern languages describe software frameworks or families of related systems" [Copl95].

At the present time, several types of patterns have been defined, such as: architectural patterns [Busc98] that show the high level architectures of a software system; design patterns [Mart98] [Mesz98] that are focused on the programming aspects, or patterns that are focused on project management [Beed97]; and patterns in agent methodology [Gior03] [Gonz04] [Gros01]. In these research works, which reflects the traditional pattern literature, a pattern is described as a tested solution to a problem.

The proposed patterns in this research work are focused on discovering the different organizational structures in the business, when an organizational plan needs to be automated. Thus, the patterns that we propose are essentially focused on discovering the different alternatives in which a process can be executed when a software system is included in the enterprise model. Therefore, a specific pattern will be used depending on the type of the organizational element to be automated. We have used methodology patterns to divide a complex problem into a specific number of solutions, where each problem is solved by a proposal pattern. In this way, the set of patterns that identifies the relevant elements to be automated is handled by a pattern language.

4.3.1 Structure of the pattern language

Several formats to represent patterns in computer science have been proposed, each one differing from the other by the kind of categories they emphasize. Among others, there is the Alexandrian form [Alex77], the GOF (Gang of Four) form [Gamm94]), and the

Coplien form [Copl95]. See [Schm95] for more examples. All formats contain the basic categories: Name, problem statement, context, description of forces, solution, and related patterns.
The basic elements for describing a pattern and its meaning in this research work are the following:
Name of Pattern: The name that identifies the pattern.
Context: A situation that address a problem. It describes situations in which the problem occurs.
Problem: The recurring problem that arises in that context. This part of a pattern description describes the problem that arises repeatedly in the given context.
Forces: Describe the relevant forces and constraints and how they interact or conflict with one another, and which goals should be achieved by implementing the solution [Url06].
Structure: A detailed specification of the structural aspects of the pattern.
Solution: Shows how to solve the recurring problem, or better, how to balance the forces associated with it.
Consequences: The benefits that the pattern provides, and any potential liabilities.
Examples of the use of the pattern.
Related Pattern: Indicate the other patterns that this pattern is composed of, is a part of, or is associated to.

4.4 Patterns in the organizational model

The pattern-oriented techniques are currently used in the solution of complex problems or in the description of a problem involving several steps.
At the present time, the pattern-based approach has been used in almost all the phases of software development. We propose a set of organizational patterns which allows us to reduce the abstraction level of an organizational model, bringing it closer to the requirements model of a software system. This is done by inserting the software system actor (SSA) into the original organizational

model and delegating the responsibilities of the organization actors to this new actor.

In this way, the proposed patterns allow us to analyze the organization elements, such as plans, resources and goals, in order to delegate these organizational behaviors to the new actor that represents the software system to be developed.

Therefore, we detected several scenarios that could exist when delegating responsibilities to the SSA. The scenarios have been grouped into five patterns that compose the proposed pattern language. It is important to point out that we adopt the definition of [Copl95] where a *pattern language* defines a collection of patterns and the rules to combine them into an architectural style.

Our proposed pattern language has been called FELRE (*From Early Requirements to Late Requirements*). The patterns that systematically guide the analyst to insert the SSA into the organizational model are the following:

- The *atomic* plan delegation pattern
- The *composite* element delegation pattern
- The *depender-dependee* element delegation pattern
- The *depender* element delegation pattern
- The *dependee* element delegation pattern

In this section, the pattern language and the set of the patterns that conforms the pattern language is explained, and a short taxonomy of the concepts used in the definition of the pattern language is shown.

4.4.1 Used Concepts

Before introducing the proposed patterns, we introduce some of the terms that will be used in the description of them. The elements have been classified according to their location in the organizational model and their characteristics; this short taxonomy can be summarized as:

a) Classification according to their location in the organizational Model:

- *Internal* elements: Are those elements that are defined inside the boundary of an organizational actor. Figure 4.2 shows an example of an internal plan. Each actor can contain several elements, which, in turn, can be subdivided into other elements. This subdivision leads a tree structure. Therefore, the internal elements can be classified according to their location in the structure hierarchy: a) the parent node could be a root node or intermediate node, and b) the child node could be an intermediate or a leaf node of the tree.
- *External* elements: Those elements that are represented in a dependency relationship as *dependum*. Figure 4.2 shows an example of an external plan.

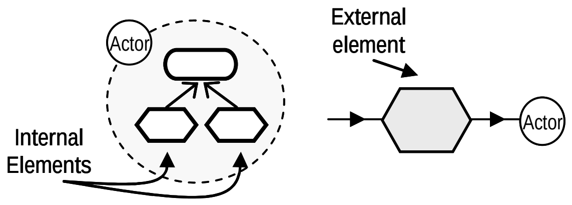


Figure 4.2 Structure of internal and external plan

b) Classification according to their location in the hierarchy structure of an actor, the elements can be:

- *Atomic* elements: Those elements that do not need to be decomposed into other sub-elements (Figure 4.3).
- *Composite* elements: Those elements whose execution is carried out by decomposing it into other sub-elements.

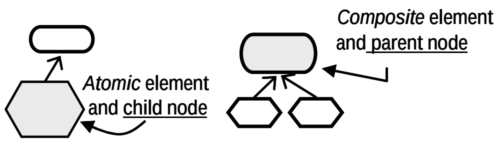
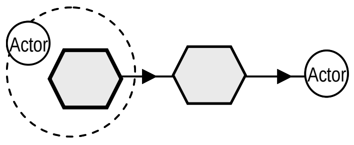

Composite element and parent node
Atomic element and child node

Figure 4.3 Example of an atomic plan and a *composite* goal

The elements explained above can also be associated to a dependency relationship; in this case, we have added the prefix *with* *dependency* to characterize them. Otherwise, the element will have the prefix *without dependency*. Examples of plans with or without a dependency relationship are explained below.

Element with dependency: Elements of this kind can not be directly performed by the element owner. Thus, other actors are needed in order to achieve this kind of modeling element. This situation is represented by associating dependencies to the plan. An example of a plan associated to a plan dependency is shown in Figure 4.4.

Element without dependencies: Elements of this kind can be directly performed by its owner. Thus, this element does not have any dependency relationship associated to it. This indicates that the achievement of the element does not require the intervention of another actor.



Plan with a dependency associated

Figure 4.4 Example of a plan with an associated dependency

All this elements will be used in the explanation of the patterns defined in FELRE pattern language.

4.4.2 The FELRE pattern language

The pattern language proposed in this thesis makes the process of insertion of the SSA in the organizational model systematic. The objective of the pattern language is to reduce the abstraction level of a *"pure"* organizational model to one closer to the requirements model. We follow the strategy of dividing the problem into more specific scenarios.

The key of the process consists in the delegation of responsibilities to the SSA. To do this, the dependencies, goals, resources and plans of the organizational actors must be redirected to the SSA. As a result of insertion process of the SSA, new dependencies need to be created in order to permit the SSA to obtain resources from the organizational actors. New dependencies also need to be created to indicate the sending of resources from the SSA to the organizational actors. The proposed pattern language must consider all the possible delegations to the SSA.

The inclusion of the software system as an actor in the organizational model allows us to have a high-level description of the plans that must be supported by the information system. This high-level description helps to focus the modeling activity on the relevant aspects to be automated, thereby, reducing the complexity of the analysis. Therefore, this model is correctly adapted to start the process of finding the requirements for the information system.

The systematic delegation of modeling elements to the SSA could cause changes in the organizational model. For this reason, to carry out the delegation process in a systematic way, it is necessary to consider all the possible scenarios in which the relevant elements can be found into the organization, and also to determine how the organizational actors interact with the elements to be delegated.

The word "*relevant*" has been used in this thesis to indicate those elements whose automatic execution satisfies the organizational goals in the most appropriate way. Therefore, once the elements to be delegated to the SSA have been selected, the process will continue analyzing the following issues: 1) the type of the element to be delegated, 2) the way in which the element is currently executed in the organizational context, 3) the way in which the plan or goals will be executed (in an automatic way) by the information system, and, 4) the roles that will be played by the original element owner once the element has been delegated to the SSA. All of these issues will be solved using the set of patterns that make up the pattern language.

Figure 4.5 shows the proposed patterns and the relationships among them, and presents a brief description of each pattern.

- The *atomic* plan delegation pattern: To be used when an *atomic plan* needs to be automated.
- The *composite* element delegation pattern: To be used when a *composite* plan needs to be automated. This pattern could be associated with the *composite* element delegation pattern. This pattern could also be associated with the following

patterns: The *depender-dependee* element delegation pattern, the *depender* element delegation pattern and *dependee* element delegation pattern.

- The *depender-dependee* element delegation pattern: To be used when both elements of the *depender* and *dependee* actors of a dependency relationship must be automated.
- The *depender* element delegation pattern: To be used when the element of the *depender* actor of a dependency relationship must be automated.
- The *dependee* element delegation pattern: To be used when the element of the *dependee* actor of a dependency relationship must be automated.

The proposed method to apply the pattern language is presented in detail in the following section.

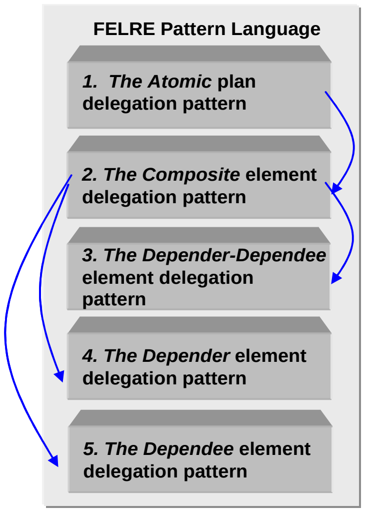

FELRE Pattern Language
1. The Atomic plan delegation pattern
2. The Composite element delegation pattern
3. The Depender-Dependee element delegation pattern
4. The Depender element delegation pattern
5. The Dependee element delegation pattern

Figure 4.5 Set of patterns of the FELRE pattern language

4.4.3 Applying pattern language

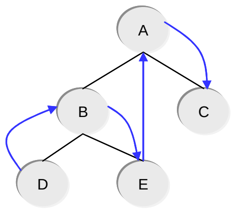
The proposed patterns must be used once the relevant elements to automate have been identified. To do this, there is a specific method to apply the proposed patterns. The application method is composed of five steps; they must be performed to correctly carry out the insertion of the SSA in the organizational model.

The steps for inserting the SSA and the elements to be automated to this actor are the following:

Step 1. Insert the SSA in the organizational model. This step concerns the insertion of a new actor which represents the software system-to-be.

Step 2. Analyze the internal elements of each actor. An analysis of the internal elements must be carried out for each organizational actor (different to the SSA). Each actor can be composed of several goals and plans, which, in turn, can be subdivided into goals or plans. As mentioned above, this leads to a tree structure. Thus, an algorithm to traverse the goal structure must be used to detect the elements that must be automated.

> **Step 2.1 Perform an in-order traversing through the internal element structures of the each organizational actor.** An in-order traversing has been proposed to analyze all the elements of the goal and plan structure tree of each organizational actor of the business. The purpose of traversing is to select the organizational elements to be automated using a software system. To perform the in-order traversing, the left tree must be analyzed first, then the parent element and, finally, the right tree. An example of the in-order traversing is shown in Figure 4.6. The tree traversing starts by analyzing the left node, then the next node to be analyzed is the parent node and later the next branch of the tree must be analyzed. Thus, in Figure 4.6, the order of the analysis in the elements should be: D, B, E, A, C. When an element to be automated is identified, the traversing in the tree must be stopped. Then, the appropriate pattern to perform the delegation must be selected. Once the element has been delegated to the SSA, the traversing process returns to the position of the relevant element in order to follow the delegation process of the elements to the SSA.

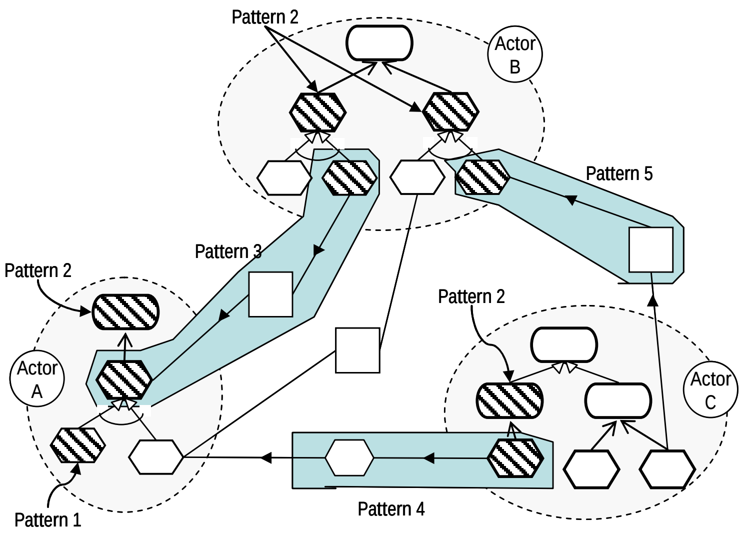


Inorder Traversing: D B E A C

Figure 4.6 Example of inorder traversing

Step 3. Identification of the appropriate pattern. Once a relevant element has been identified (in the previous step), it is necessary to identify the pattern type that corresponds to the analyzed element.
The next sub-steps detail this process. For example, in Figure 4.7, the following elements to be automated have been depicted with a background of parallel lines, applying the steps mentioned above, if the actor "*A*" is analyzed, then the first relevant element found is an *atomic* plan (because this is not decomposed into other *subplans*). Thus, the pattern to be used to perform the delegation of this element to the SSA is the *atomic* plan delegation pattern. Figure 4.7 shows the appropriate pattern to be used for each element in the model. The patterns are identified in figure by the pattern number (pattern 1: The *atomic* plan delegation pattern, pattern 2: The *composite* element delegation pattern, pattern 3: The *Depender-Dependee* element delegation pattern, pattern 4: The *Depender* element delegation pattern, pattern 5: The *Dependee* element delegation pattern).

Step 3.1 Analysis of elements not associated to dependency relationships. When the analyzed element plan is placed in an *end* node of the tree, then the pattern used for this element is the *atomic* plan delegation pattern (pattern 1). However, when the element is a plan or a goal which is decomposed into other *subplans* needed to execute it, the pattern to be used is the *composite* plan delegation pattern. Figure 4.7 shows the actor "*B*" as an example of this kind of the pattern (Pattern 2).



▨ Elements to be delegated

Figure 4.7 Identification of patterns in an organizational model

Step 3.2 Analysis of elements associated to dependency relationships. When the *dependum* object in this relationship is a plan or a resource, the other element that is joined to the dependency must be analyzed. If the element needs to be delegated to the SSA, the pattern used for these elements is the *depender-dependee* element delegation pattern (to see Figure 4.7, pattern 3).

However, if only one element of the dependency relationship must be delegated, the role played by the organizational actor that contains the element must be analyzed.

If the role of the actor in the dependency is *depender*, the pattern used for this element is the *depender* element delegation pattern (to see Figure 4.7, pattern 4). Otherwise, the *dependee* element delegation pattern must be used to see Figure 4.7, pattern 4.

Step 4. Delegate the relevant elements to the SSA. The steps described in the appropriate pattern must be followed in order to carry out the delegation of the element (s) to the SSA.
Step 5. Following with the inorder traversing. Once the pattern for a relevant element has been identified, the analysis in the internal elements of an actor must continue until all the elements of the actor had been analyzed. In this case, the analysis will continue with the next actor represented in the organizational model.

4.4.4 Catalog of Patterns

In this section, each pattern of the pattern language proposed is explained in depth. All these patterns concern the delegation of plans from the organizational actors to the SSA; this delegation process depends on three issues:

- The type of the plan to be delegated
- How the plan is currently executed in the organizational context
- How the plan will be executed in an automatic way

The structure used to detail each pattern is the following:
Name: The name of each proposal pattern must represent its objective and it's the intended meaning (as much as possible). For example, in the *atomic* plan delegation pattern, the name makes reference to the type of element that is to be delegated to the SSA. Additionally, the name of each pattern has the word *delegation,* which indicates that the objective of the pattern is the delegation of the element to the SSA.
Context: In this section, the situations in which the problem occurs are explained. This section details the initial situation in the business before the pattern is applied to it.
Problem: In this section, the reasons why the pattern is being developed are presented. We have divided a complex problem (modeling an organization with the system-to-be, where the execution of the plans will be modified by a new actor (SSA) in several sub problems. We lead each specific problem with a specific pattern and explain the forces that influence the solution of the problem.

Structure: In this section, the structural aspects of each pattern are shown. This section also illustrates the typical scenarios of the behavior of this pattern in the organizational model. We graphically illustrate each scenario.
Solution: In this section, the pattern solution is detailed in a set of steps, which provide a correct implementation of the solution to each problem.
Example: In order to illustrate each pattern, the *Car Rental* case study has been used. This is a real project of the Care Technology Company, which concerns organizational modeling for a car rental enterprise in Alicante, Spain. We explain each pattern showing the initial context in which the problem emerges; we show how the selected pattern is applied, and, finally, we present the transformation steps for creating a new context.
Related Pattern: in this section, we indicate the situation where a pattern is associated to another pattern. A pattern solves a particular problem, but its applications may address new problems. Some of these can be solved by other patterns [Busc98].
The proposed patterns are detailed in the next sub-sections. First, a brief summary to describe the pattern is shown, and second, all the elements of the pattern are detailed.

4.4.4.1 The *atomic* plan delegation pattern

This pattern must be used when an *atomic* plan needs to be delegated to the SSA in order to automate its execution. The *atomic* plans are those plans that do not need to be divided into other *subplans* to be executed.
The pattern details the problems that may be found in the delegation of an *atomic* plan and also shows the alternative solutions for that delegation. Figure 4.8 presents an example of this structure.

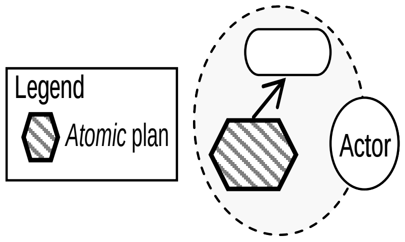


Figure 4.8 Example of the atomic plan

4.4.4.1.1 Context

This pattern concerns the delegation of an *atomic* plan to the SSA, which must fulfill the following conditions:

- It is not decomposed into other *subplans* and
- It is not associated to any dependency relationship.

Figure 4.9 illustrates an example of an *atomic* plan in the *Car Rental* case study. Specifically, the figure represents the plan: *provide info of prices* of the *employee* actor. This plan has been selected to be automated. For this reason; the plan needs to be delegated to the SSA.

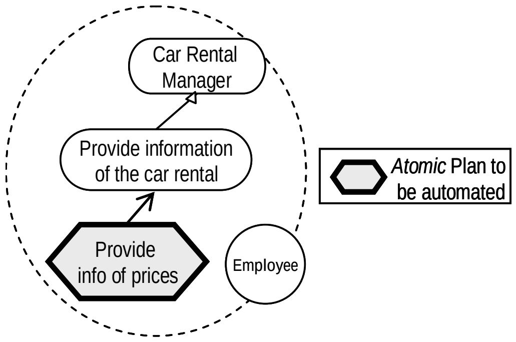

Car Rental Manager
Provide information of the car rental
Provide info of prices
Employee
Atomic Plan to be automated

Figure 4.9 An example of an atomic plan in the *Car Rental* case study

4.4.4.1.2 Problem

The problem consists of determining the role played by the original plan owner once the atomic plan has been delegated to the SSA. It is also necessary to determine the influence of this delegation on the other organizational actors involved in the business.

4.4.4.1.3 Forces

There are three forces associated to the solution of this pattern:

- The *atomic* plan to be automated needs the intervention of the original owner actor.
- The *atomic* plan to be automated needs the intervention of the original owner actor, as well as the intervention of other organizational actors.
- The *atomic* plan to be automated doesn't need the intervention of any organizational actor.

4.4.4.1.4 Structure

The elements used in this pattern are the following:

- **A*tomic* plan:** This is the element that needs to be delegated to the SSA.
- **An organizational actor:** This actor, who is the original *atomic* plan owner, can play the role of the *depender* or *dependee* actor once the plan has been delegated.
- **A parent node:** This is the element linked to the *atomic* plan. This element can be another plan or a goal.
- **A link:** This element joins the *atomic* plan with its parent node.

Note that, in this kind of pattern, the *atomic* plan does not have any dependency relationship.

Scenarios:

There are three possible scenarios in which an *atomic* plan to be automated can be found in the organizational context:

Scenario I. This describes the situation where the *atomic* plan is associated to its parent plan by an *AND* decomposition link. Figure 4.10 (a) depicts this situation.

Scenario II. This describes the situation where the *atomic* plan is associated to its parent plan by an *OR* decomposition link. Figure 4.10 (b) depicts this situation.

Scenario III. This describes the situation where the *atomic* plan is associated to its parent node (a goal) by a *means-end* link. Figure 4.10 (c) depicts this situation.

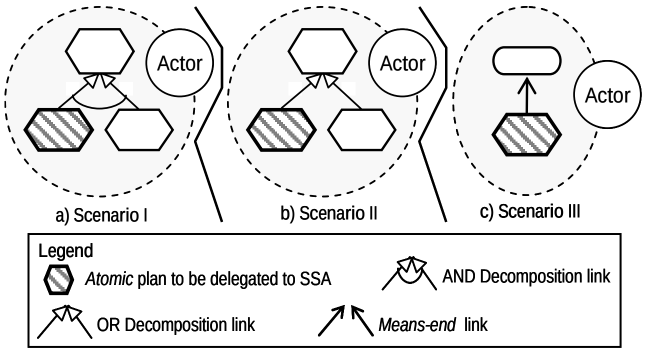


Figure 4.10 Scenarios of an atomic plan into an organizational model

4.4.4.1.5 Solution

The process to delegate an *atomic* plan to the SSA consists of four steps:

Step 1. Delegate the analyzed *atomic* plan to the SSA.

Step 2. Determine the roles that the organizational actor (who was responsible for this plan) will play after the plan is delegated to the SSA. These roles and their solutions are described in the following sub-steps:

> **Step 2.1** If the original plan owner will play the role of *Provider* of information to perform the plan (once the plan has been delegated), then a resource dependency between the actor and the SSA must be created, in order to indicate the introduction of information to the software system from the organizational actor. The *depender* of this dependency will be the SSA and the *dependee* will be the original plan owner. The application of the solution implies the analysis of the plan name; it must be changed so that it is more appropriated, for the intended semantics.
>
> **Step 2.2** If the original plan owner will play the role of *Requester* of information (once the plan has been delegated), then a resource dependency between the actor and SSA must be created. The *depender* of this dependency will be the original plan owner and the *dependee* will be the SSA.

This new dependency indicates the delivery of information to the organizational actor.

Step 2.3 If the original plan owner does not have any interaction with the SSA to perform the plan, no dependencies must be created. The selection of this alternative implies the analysis of the plan name in order to make it appropriate for the new organizational configuration.

Step 3. Determine the role that the other organizational actors play in the delegated plan. If they want to obtain or to provide information for the plan, then, new dependencies among these actors and the SSA must be created.

Step 4. If more than one dependency relationship is generated during the delegation of an *atomic* plan to SSA, then, they must be labeled with the same number in order to indicate their association.

Step 5. Analyze the context of the atomic plan. In this step, the atomic plan must be analyzed in the context of its hierarchical structure, in order to determine if its parent goal must also be automated. In this specific case, the *composite* plan delegation pattern must be used.

4.4.4.1.6 Examples

The application of the steps of the *atomic* plan delegation pattern is illustrated with the example shown in Figure 4.9.

The first step in the solution of this pattern consists of delegating the plan *provide info of prices* of the *employee* actor to the *Car Rental* actor.

The second step in the solution of this pattern consists of determining the role played by the *employee* actor. The different roles that the *employee* actor can play are shown below.

The first alternative solution for the second step must be applied when the organizational actor plays the role of *Provider* of information for the plan delegated to the SSA. This decision implies changing the plan name to make it more appropriate for the new configuration. The *Car Rental* case study, the *employee* actor acts as *Provider* of information. Thus, the plan *provide info of prices* should be changed *Calculate prices* (Figure 4.11), because the *employee* actor will provide the information about the prices of a reservation. In this case, a new resource dependency is created between the *employee* actor and the SSA, where the *depender* actor will be the *Car Rental* actor.

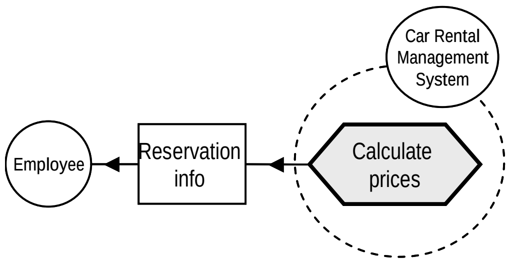


Figure 4.11 An example when the employee actor acts as *provider* of information)

The second alternative solution for the second step must be applied when the actor (who was responsible for the delegated plan) plays the role of *Requester* of information. For example, if the *employee* actor (Figure 4.9) acts as *Requester* of information, then the *employee* actor will handle the software system to obtain the prices of the reservation. Therefore, a resource dependency *(prices and models info*) between the *employee* and the *Car Rental* actor must be created. This new dependency will indicate the delivery of information of the *Car Rental* actor to the *employee* actor. Figure 4.12 depicts this example.

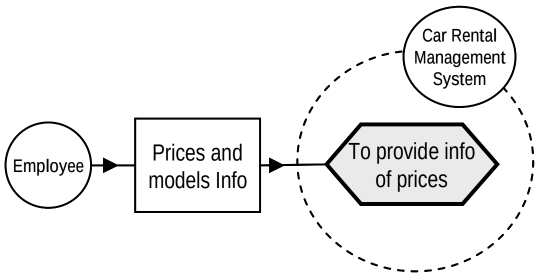


Figure 4.12 An example when the employee actor acts as *requester* of information)

The third alternative solution for the second step must be applied when the actor (who was responsible for the delegated plan) does not have any interaction with the SSA.
For example, the plan *To provide information of prices* (Figure 4.9) that has been delegated to the *Car Rental* actor does not require any interaction with other organizational actors in order to be executed. Therefore, the original name of the plan (*To provide info of prices)* must be modified to represent the fact that this plan will be executed by the *Car Rental* actor itself. The new name of this plan is: *Calculate prices* as shown in Figure 4.13.

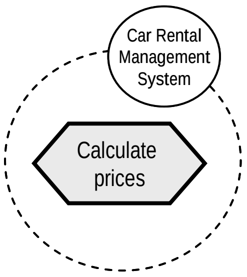


Figure 4.13 An example when the employee actor does not have any interaction with the delegated plan

The third step in the solution of this pattern consists of creating new dependencies among the organizational actors, and the *Car Rental* actor must be created if other organizational actors want to obtain or provide information about the delegated plan. The Figure 4.14 depicts an example, where the *associated branches* actor provides info to the *Car Rental* actor through a resource dependency (*Prices and models info*). Thus, only this new dependency is created in this step.
The fourth step in the solution of this pattern consists of labeling the dependency relationship generated during the delegation of an *atomic* plan to the SSA in order to indicate the association between them. For example, in Figure 4.14, the two dependencies created in the delegation process can be labeled with the number 1.
The fifth step in the solution of this pattern consists of analyzing the context of the original *atomic* plan (*To provide info of prices*) in

Figure 4.9 in the analyzed example, the plan is linked to a goal by a *means-end* link; therefore, the ***composite*** plan delegation pattern must be used to analyze this situation. An example of delegating a *composite* plan is shown in the following pattern.

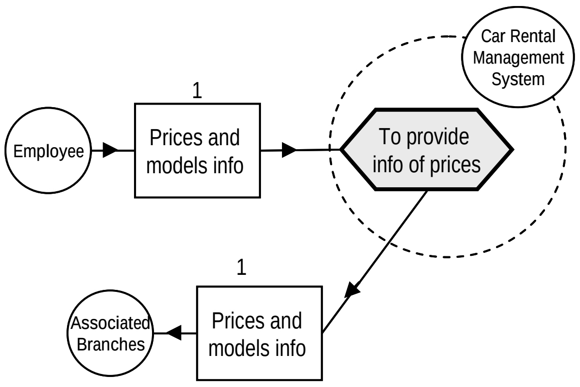


Figure 4.14 An example when other actors have interaction with the delegated plan

4.4.4.2 The *composite* element delegation pattern

This pattern must be used when a *composite* element needs to be delegated to the SSA in order to automate its execution. The node can be a goal or a plan. Figure 4.15 depicts an example of this structure, where a *composite* plan is linked to its children nodes by an *OR* decomposition link. The pattern details the problems of this action, and shows the alternative solutions for these problems.

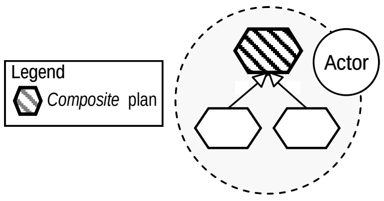


Figure 4.15 Example of a *composite* plan

4.4.4.2.1 Context

This pattern concerns the delegation of a *composite* element to the SSA. It can be a plan or a goal, which must fulfill the following conditions:

- If the *composite* element is a plan, then it must be decomposed into other *subplans*.
- If the *composite* element is a goal, then it must be composed only by *subplans* through a *means-end* link.
- At least one *subplan* of the *composite* element must have been delegated to the SSA.

Note that, a *composite* element can have a dependency relationship associated to it. The diagram in Figure 4.16 illustrates an example of a *composite* goal in the *Car Rental* case study; specifically in *analyze availability in another branch.* This goal must be analyzed to be delegated to the SSA.

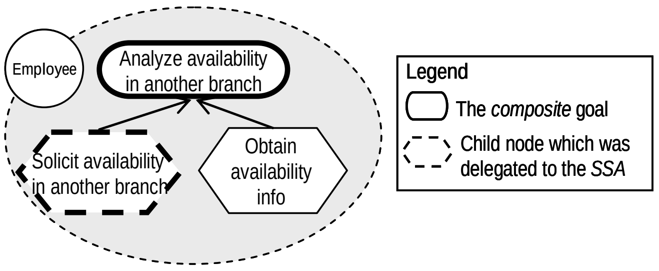

Employee
Analyze availability in another branch
Solicit availability in another branch
Obtain availability info
Legend
The composite goal
Child node which was delegated to the SSA

Figure 4.16 An example of a *composite* goal in the *Car Rental* case study

4.4.4.2.2 Problem

The problem consists of determining when a *composite* element must be delegated to the SSA. The way the delegation influences in its *subplans* and the organizational actors must also be analyzed.

4.4.4.2.3 Forces

There are three forces associated to the solution of this pattern:

- The *composite* element has at least one *subplan*, which must have been delegated to the SSA.
- The *composite* element to be delegated needs the intervention of the same organizational actor.

- The *composite* element has a dependency with another organizational actor.

4.4.4.2.4 Structure

The pattern is composed of the following elements:

- **A *composite* element:** this element can be a goal or plan which requires to be delegated to the SSA.
- **Child nodes:** these elements must be plans.
- **Links:** this element joins the *composite* plan with its child nodes.
- **An organizational actor:** is the actor who contains to original *composite* element to be delegated. This actor could have an interaction with the *composite* element once the element has been delegated by a dependency relationship.
- **A dependency relationship:** this element is optional in this pattern. The type of the dependency of our interest could be: resource dependency, plan dependency or goal dependency.

Scenarios:
There are three possible scenarios in which a *composite* element can be found in the organizational context:

Scenario I. This describes the situation where a *composite* plan is associated to its child nodes by an *AND* decomposition link. Figure 4.17 (a) depicts this situation.

Scenario II. This describes the situation where a *composite* plan is associated to its child nodes by an *OR* decomposition link. Figure 4.17 (b) depicts this situation.

Scenario III. This describes the situation where a *composite* goal is associated to its child nodes by a *means-end* link. Figure 4.17 **(**c) depicts this situation.

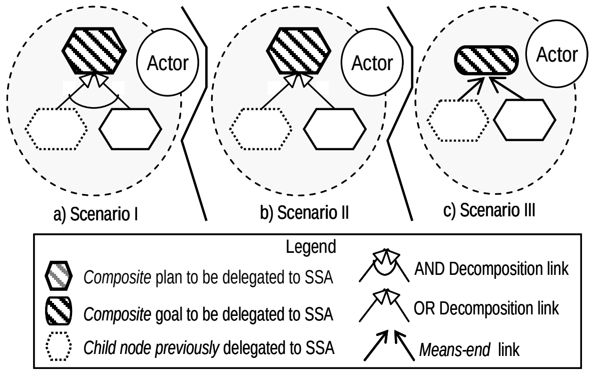


Figure 4.17 Scenarios of a *composite* element into an organizational model

4.4.4.2.5 Solution

The process of delegating a *composite* plan to the SSA is influenced by the previous delegation of at least one child node to the SSA. The *composite* goal will only be considered for delegation to the SSA, if it is linked with its child nodes (plans) by a *means-end*. This process is composed of five steps:

Step 1. Analyze the *composite* element to determine if it can be delegated to the SSA.

> **Step 1.1** When the *composite* element is a plan, its nodes must be analyzed if at least one child node of the *composite* plan was delegated to the SSA. If this condition is satisfied, then the *composite* plan must be delegated to the SSA.
>
> **Step 1.2** When the *composite* element is a goal, several conditions must be taken into account to delegate the goal to the SSA. 1) The children nodes of the *composite* goal must be plans, and they must be linked by *means-end* links, and 2) At least one child node of this goal must have been delegated previously to the SSA. If these two conditions are satisfied, then the goal can be delegated to the SSA.

Step 2. Delegate the *composite* element to the SSA if it satisfies the conditions explained in step 1.2.

Step 3. Associate the *subplans* of the *composite* plan/goal located in the SSA. The link used to associate these elements must be the same link that the *composite* plan/goal had before being delegated to the SSA.

Step 4. Analyze the influence of this delegation on the organizational actors. This influence only occurs when the *composite* element is a plan.

> **Step 4.1** If an organizational actor provides information to the *composite* plan to execute the plan, a resource dependency between the actor and the SSA must be created. The *depender* of this dependency is the SSA. The new dependency indicates the reception of information from the organizational actor to the SSA.
>
> **Step 4.2** If an actor requires information from the *composite* plan, a resource dependency between the actor and the SSA must be created. The *depender* of this dependency is the organizational actor and the *dependee* is the SSA. The new dependency indicates the delivery of information to the organizational actor from the SSA.
>
> **Step 4.3** If the delegation of the *composite* element does not affect any actor because there is no direct interaction with the element, then no dependency relationship between the organizational actors and the delegated element is created.

Step 5. Determine whether the *composite* plan/goal to be delegated to the SSA has a dependency associated to it. In this case, it is necessary to determine if an associated pattern must be applied. The patterns that can be applied are: the *depender-dependee* element delegation pattern or the *depender* element delegation pattern.

4.4.4.3 The depender-dependee element delegation pattern

This pattern must be used when all the elements of a dependency relationship (*depender-dependum-dependee*) need to be delegated to the SSA. In other words, the element of the *depender* actor as well as

the element of the *dependee* actor must be executed in an automatic way.
The pattern details the problems that may be found in the delegation of the elements of the *depender*-*dependee* actors and shows alternative solutions. Figure 4.18 depicts an example of this structure.

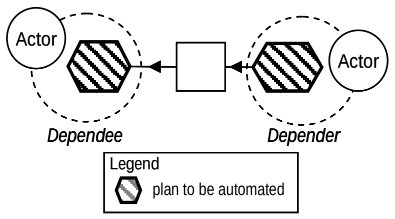


Figure 4.18 Example of the depender actor plan and the dependee actor plan to be automated

4.4.4.3.1 Context

This pattern concerns the automation of the elements of the *depender* actor and the *dependee* actor, where the elements to be delegated are associated by a dependency relationship. To apply this pattern, the following conditions must be fulfilled:

- The elements of the organizational actors associated by the dependency relationships need to be delegated to the SSA; these elements can be a goal or a plan,
- The *dependum* object must be a resource or a plan

Figure 4.19 illustrates an example of this pattern, where the plans of the *depender* and *dependee* actors must be delegated to the SSA. The delegation of these elements focuses on the *dependum*, which is a resource (*Customer info*). Therefore, both, the acquisition as well as the delivery of this resource, need to be automated.

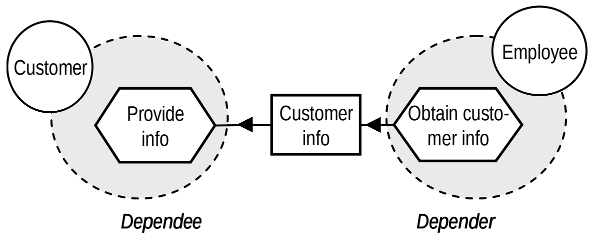


Figure 4.19 An example of the depender-dependee element delegated pattern in the *Car Rental* case study

4.4.4.3.2 Problem

The delegation of the elements of the *depender* and *dependee* actors causes several changes in the entire organizational context; mainly, in the actors involved in the dependency relationship. These changes are related to the type of elements that compose the dependency. These changes also depend on the role played by the actors involved in the dependency relationship analyzed.

4.4.4.3.3 Forces

There are five forces associated to the solution of this pattern:

- The elements of the *depender*/*dependee* actors to be automated are linked to a resource dependency (*dependum*).
- The elements of the *depender*/*dependee* actors to be automated are linked to a plan dependency (*dependum*).
- The element of *depender* actor is a goal, and both the element of *dependee* actor and the *dependum* are plans.
- The plans delegated to the SSA require the intervention of the original owner actors.
- The plans delegated to the SSA require the intervention of other organizational actors, not just the original owner actors.

4.4.4.3.4 Structure

The elements used in this pattern are the following:

- **Organizational actors:** these are the *depender* actor and the *dependee* actor in the analyzed dependency relationship.
- ***Depender* actor element:** this is the element that needs to be delegated to the SSA. The element must be a plan or a goal.

- ***Dependee* actor element:** this is the element that needs to be delegated to the SSA. The element must be a plan or a goal.
- ***Dependum*:** this element represents the context around of the dependency; it can be a resource or a plan.
- **Dependency relationship:** this element joins the *depender/dependee* actors and the *dependum* object.

Scenarios:
There are three possible scenarios in which this pattern can be found in the organizational context:
Scenario I. This describes the situation where both the element of the *depender* actor and the element of the *dependee* actor are plans, in which case they must be delegated to the SSA, and where the *dependum* object is a resource. Therefore, the scenario represents the need of the business to obtain and to send a resource in an automatic way. Figure 4.20 (a) depicts this situation.
Scenario II. This describes the situation where both, the element of the *depender* actor and the element of the *dependee* actor are plans which must be delegated to the SSA, and where the *dependum* object is a plan. Therefore, the scenario represents the need of the business to automate the *depender* plan which has been delegated to another actor. Figure 4.20 (b) depicts this situation.
Scenario III. This describes the situation where both, the *depender* element is a goal and the *dependee* element is a plan. These elements must be delegated to the SSA. The *dependum* object is a plan. Therefore, this scenario represents the delegation of a goal to the SSA, which delegates the execution of a plan to another actor. Figure 4.20(c) depicts this situation.

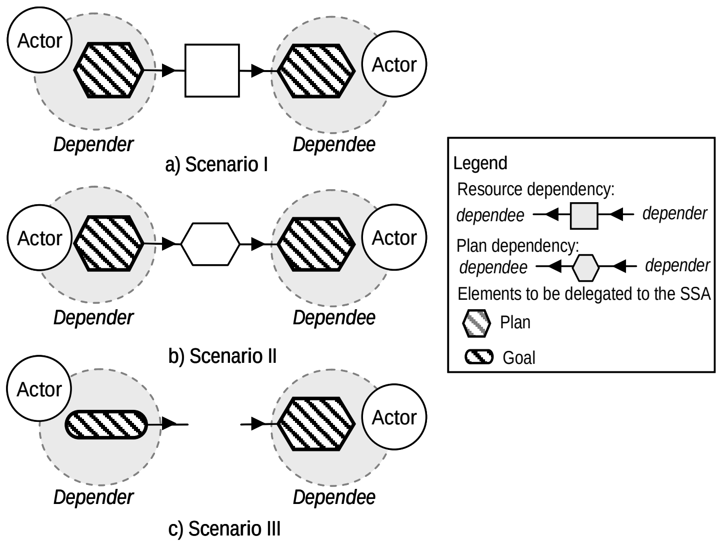


Figure 4.*20* Scenarios of the depender- dependee element delegation pattern

4.4.4.3.5 Solution

The delegation of the elements of *depender* and *dependee* actors to the SSA focuses on the following issues: a) the roles played by the organizational actors, b) the type of the elements involved in the dependency relationship, and c) the type of the *dependum*. Therefore, the alternative solutions are classified depending on the elements to be delegated.

a) First alternative: Plan-Resource-Plan,
b) Second alternative: Plan-Plan-Plan,
c) Third alternative: Goal-Plan- Plan

The first element indicates the *depender* actor; the second element indicates the *dependum*, and the third element belongs to the *dependee* actor.

a) First Alternative (Plan-Resource-Plan):

The first alternative is used when both, the *depender* and the *dependee* plan must be delegated to the SSA, and the *dependum* object is a resource. This indicates the need to automate the sending and receiving of the resource. The first alternative of solution is done in four steps:

Step 1. Delegate both the *depender actor plan as well as the dependee* actor plan to the SSA, and place a *composite* plan, which joins these plans through an *AND* in the SSA. Figure 4.21 illustrates the delegation of the plans of both, the *depender* and *dependee* actors, to the SSA. The plans are placed as child nodes of a *composite* plan, which must be created in order to determine the association between the two plans. The plans and the goal will be joined by an *AND* link.

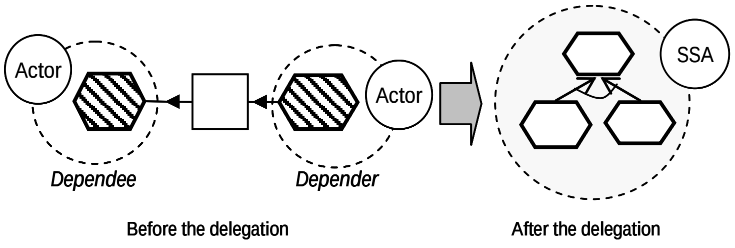


Figure 4.21 Before and after delegating the plans of the depender/dependee actors to the SSA

Step 2. The original resource dependency between the organizational actors must be redefined. The *depender* actor of the new dependency will be the SSA, and, the plan associated to the dependency will be the plan that needs the resource to be performed. The selection of the *dependee* actor in the relationship will depend on which actor acts as *Provider* of information to perform the plan.

Step 2.1 If the actor who acts as depender in the dependency relationship analyzed (that we called *O-Der*) will play the role of *Provider* of information to execute the plan, then the original dependency between the *O-Der* actor and original dependee actor (*O-Dee*) remains the same and a new dependency between the SSA and *O-Der* actor is created (Figure 4.22). These dependencies indicate that the SSA depends on the organizational actor to obtain the information required to execute the plan.

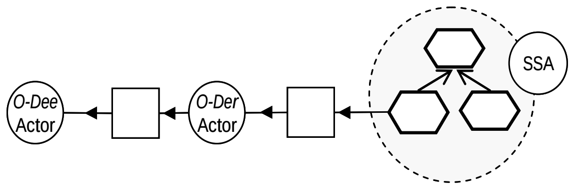


Figure 4.22 Organizational model after applying step 2.1 (the O-Der actor acts as *provider* of information)

Step 2.2 In contrast to step 2.1, if the original dependee actor (O-Dee) actor will play the role of *Provider* of information to execute the plan, then the original resource dependency is redefined between the SSA and the O-Dee actor. The SSA will act as depender. Figure 4.23 shows the resource dependency where the depender actor is the SSA and the dependee actor is the same of the original dependency.

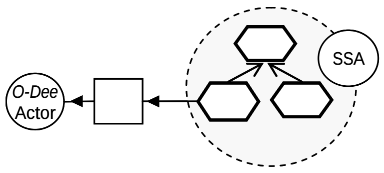


Figure 4.23 Organizational model after applying step 2.2 (the O-Dee actor acts as *provider* of information)

Step 2.3 If both organizational actors need to interact with the SSA, the original resource dependency will be redefined between the actor that acts as *Provider* and the SSA. A new dependency must also be created between the other organizational actors and the SSA. Figure 4.24 shows the alternatives where both organizational actors need to interact with the SSA. Therefore, the original resource dependency is redefined between the actors that act as *Provider*; in this case, the SSA will act as depender. When the organizational actor acts as *Requester,* a new resource dependency will be placed between the organizational actor and the SSA. The dependee actor will be the SSA.

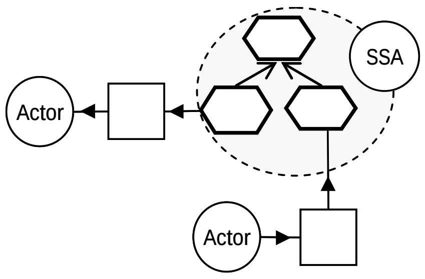


Figure 4.24 Organizational model after applying step 2.3 (both actors interact with the SSA)

Step 2.4 If no actor has any interaction with the SSA to execute the delegated plans, no dependencies must be created. The selection of this alternative implies the analysis of the plan name in order to make it appropriate for the new organizational configuration.

Step 3. Analyze the influence of the delegation of the elements of the *depender and dependee* actors on the organizational actors. When other organizational actors must obtain or provide information from/to the delegated plans, new dependencies among these actors and the SSA must be created. If there is an interaction between the organizational actors (*O-Der* and *O-Dee*), a new dependency between the actors must be created.

Step 4. If more than one dependency relationship is generated during the delegation of elements of the *depender/dependee* actors to the SSA, they must be labeled with the same number, in order to indicate their association.

b) Second Alternative (Plan-Plan-Plan):

The second alternative is used when both the *depender* and the *dependee* plan must be delegated to the SSA, and the *dependum* object is a plan. This indicates the delegation of a plan of the *depender* actor to another actor who will act as *dependee*. The automation of these plans will be carried out as follows:

Step 1. Delegate the *depender* actor plan to the SSA, and place the *dependee* actor plan as a *subplan*. These plans will be linked by an *AND* decomposition link. Figure 4.25 illustrates the delegation of the plans of both the *depender* and the *dependee* actors to the SSA. The plan of the *depender* actor is placed as parent node of the *dependee* actor plan. These plans are joined by an *AND* decomposition link.

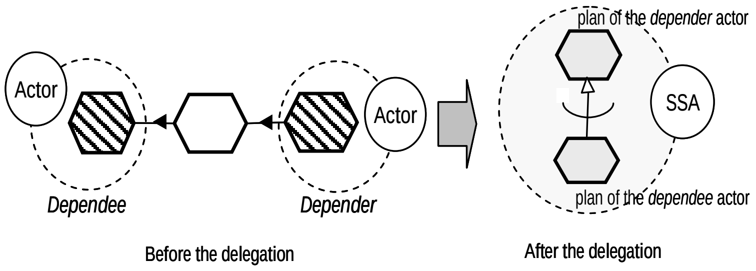


Figure 4.25 Organizational model before and after applying the step 1 of the second alternative

Step 2. Determine the roles played by the organizational actors with the delegated plans.

Step 2.1. If an actor plays the role of *Provider* of information in some of the delegated plans to the SSA, a resource dependency between this actor and the SSA must be created.

The SSA will act as depender in this dependency relationship. This new dependency indicates the delivery of information by the SSA.

Step 2.2. If some actor plays the role of *Requester* of information in some of the delegated plans to the SSA, a resource dependency between the actor and SSA must be created. The SSA will act as dependee in this dependency relationship. This new dependency indicates the delivery of information to the organizational actor.

Step 2.3. If the actor does not have interaction with the SSA to perform the plans delegated, no dependencies must be created. The selection of this alternative implies the analysis of the plan name in order to make it appropriate for the new organizational configuration.

Figure 4.26 illustrates an example of this alternative, where actor 1 acts as *Provider* of information with the delegated plan to the SSA. A resource dependency is used to model this option, and the SSA acts as *depender* in this relationship. Actor 2 acts as *Requester* of information so another resource dependency is placed in the model. The SSA acts as *dependee* in the dependency relationship.

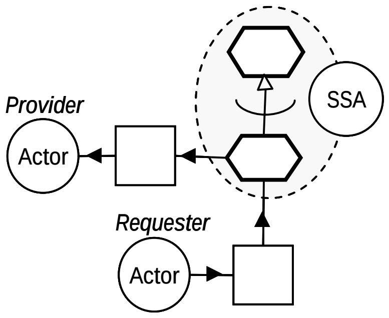


Figure 4.26 Organizational model after applying step 2 of the second alternative

Steps 3 and 4 of the first alternative must be taken into account in order to carry out all the processes for delegating the elements of the pattern to the SSA.

c) Third Alternative (Goal-Plan-Plan):

The third alternative is used when the elements of a dependency relationship are: a goal in the *depender* actor, a plan in the *dependee* actor, and a plan as *dependum*. It indicates the need to execute a plan for another actor to achieve a goal. Therefore, the delegation of these elements is carried out as follows:

Step 1. Delegate the *depender* actor goal to the SSA and place the dependee actor plan as its child node. These elements will be linked by a *means-end* link. Figure 4.27 illustrates the delegation of the *depender* actor goal and the delegation of the plan of the *dependee* actors. The goal is placed as parent node, and the plan is placed as the child node of this goal. These elements are joined by a *means-end* link.

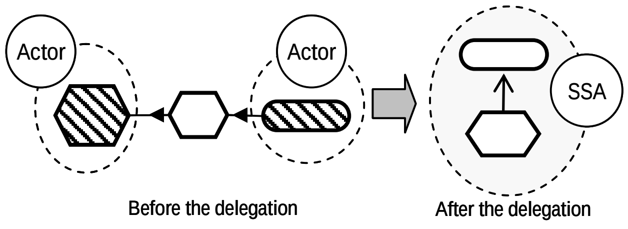


Figure 4.27 Organizational model before and after to apply the step 1 of the third alternative

Step 2. Determine the roles played by the organizational actors with the delegated plans.

Step 2.1. If an actor plays the role of *Provider* of information in some of the delegated plans to the SSA, a resource dependency between this actor and the SSA must be created.

The SSA will act as depender in this dependency relationship. This new dependency indicates the delivery of information by the SSA.

Step 2.2. If some actor plays the role of *Requester* of information in some of the delegated plans to the SSA, a resource dependency between the actor and SSA must be created. The SSA will act as dependee in this dependency relationship. This new dependency indicates the delivery of information to the organizational actor.

Step 2.3. If the actor does not have interaction with the SSA to perform the plans delegated, no dependencies must be created. The selection of this alternative implies the analysis of the plan name in order to make it appropriate for the new organizational configuration.

Step 3. Analyze the influence of the delegation of the elements of the *depender and dependee* actors on the organizational actors. When other organizational actors must obtain or provide information from/to the delegated plans, new dependencies among these actors and the SSA must be created. If there is an interaction between the organizational actors (*O-Der* and *O-Dee*), a new dependency between the actors must be created.

Step 4. If more than one dependency relationship is generated during the delegation of elements of the *depender/dependee* actors to the SSA, they must be labeled with the same number, in order to indicate their association.

Figure 4.28 shows the final model of the example illustrated in Figure 4.27. The delegation of the *depender* actor element and the *dependee* actor element are a goal and a plan. These elements are joined by a *means-end* link.

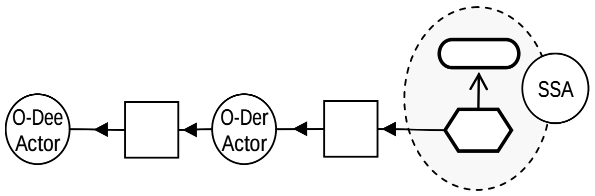


Figure 4.28 Organizational model after applying steps from the third alternative of the depender-dependee element delegation pattern

The *O-Der* actor acts as *Provider* of information to the SSA. A resource dependency is used for modeling this option, and the SSA will act as *depender* actor in this relationship. There is an interaction between the organizational actors which is depicted through a resource dependency between the *O-Dee* actor and the *O-Deer* actor.

4.4.4.3.6 Examples

The delegation of the elements of *depender* and *dependee* actors to the SSA is classified depending on the elements to be delegated. In this sub-section we give an example for each alternative of delegation, they are:

- First alternative: Plan-Resource-Plan,
- Second alternative: Plan-Plan-Plan,
- Third alternative: Goal-Plan- Plan

Example of the first Alternative (Plan-Resource-Plan):

The application of the steps of this alternative is illustrated with the example shown in Figure 4.20, where both actors *depender* and *dependee* want to delegate their plan to the SSA. They are: *Obtain Customer info* and *Provide info.* Thus, the first steps in the solution of this pattern consist of delegating both plans to the SSA, and place a plan joined through an *AND link* in the SSA. Next step consists in redefining the *dependum* element (resource: *Customer info*). The *depender* actor in the dependency will be the plan which needs the resource; meanwhile, the *dependee* actor in the relationship will be the actor who acts as *Provider* of information to perform the plan. Specifically, In this case the *employee* actor acts as *Provider* of information to execute the plan, then the resource dependency remains the same and a new dependency between the SSA and *Employee* actor is created. In step 4, all the dependencies modified in these alternatives are labeled with the number 1. Figure 4.29 shows the result of applying this alternative of solution.

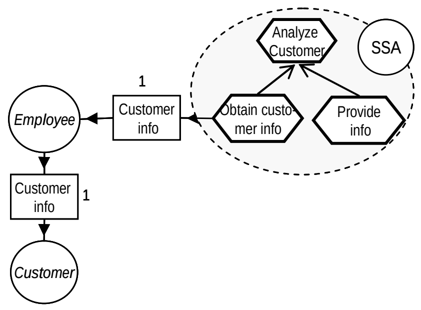


Figure 4.29 Organizational model after applying steps of the pattern

Second Alternative (Plan-Plan-Plan):

The application of the steps of the second alternative is illustrated with the example shown in Figure 4.30, where both actors *depender* and *dependee* want to delegate their plan to the SSA (*Register car reservation* and *Send info*); also the *dependum* object is a plan (*Send info*).

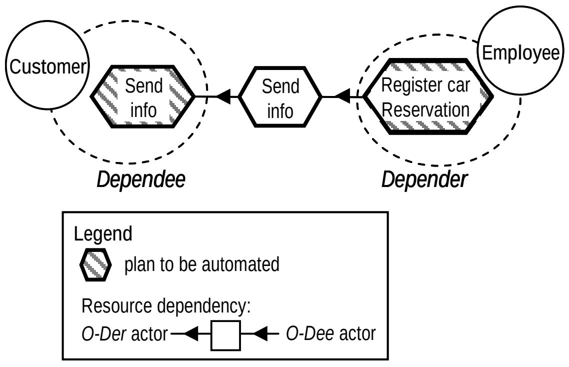


Figure 4.30 Example of second alternative

First step in the solution of this pattern consists of delegating both plans to the SSA, and place the *dependee* actor plan as a *subplan* of the *Register car reservation* plan.

The delegation of the *dependee* actor plan (*Send info)* implies change in the name of the *subplan*, in order to have a more appropriate name for the intended semantics. Thus, the plan in this case is *Obtain information of the reservation*. These plans will be linked by an *AND* decomposition link.

The second step is determining the role played by each actor, in this case, the two actors *Customer* and *Employee* can carry out the register of a car reservation, and namely they act as *Provider* of information with the SSA. Therefore, a resource dependency between these actors and the SSA is created. The resource dependencies are labeled with the number 2 in order to indicate their association.

Figure 4.31 depicts this alternative, where the plans of the *Customer* and *Employee* actors have been delegated to the SSA. The example shows also the new resource dependencies generated in this alternatives, it is because both actors act as *Provider* of information of the plan (*Obtain information of the reservation*).

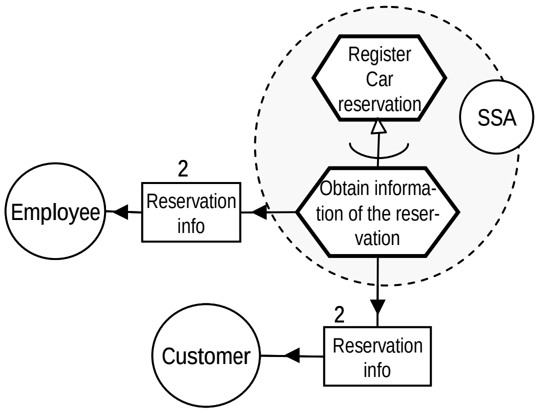


Figure 4.31 Example of second alternatives in the *Car Rental* case study

4.4.4.4 The *depender* element delegation pattern

This pattern must be used only when the element of the *depender* actor needs to be delegated to the SSA in order to automate its execution; this element can be a plan or a goal, and where the *dependum* object can be a resource or a plan. The pattern details the problems that may be found in the delegation of the *depender* element actor and also shows the alternative solutions for that delegation. Figure 4.32 depicts an example of this structure.

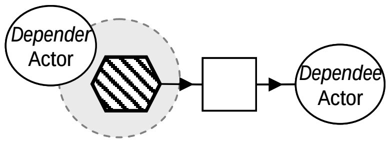


Figure 4.32 Example of a depender actor plan to be automated

4.4.4.4.1 Context

This pattern concerns the automation of the element of the *depender* actor, where the analyzed element has associated a dependency relationship. Two conditions must be fulfilled in order to delegate the analyzed element:

- An element of the *depender* actor needs to be delegated to the SSA; this element can be a goal or a plan,
- The *dependum* object must be a resource or a plan

Figure 4.33 illustrates an example of this pattern, where the plan (*Obtain date and model car)* of the *depender* actor must be delegated to the SSA. The delegation of this plan focuses on the *dependum,* which is a resource (car *info*). Therefore, the acquisition of this resource needs to be automated.

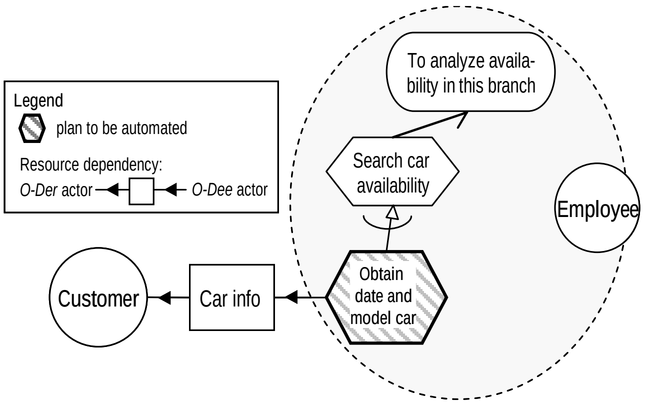


Figure 4.33 An example of the depender element delegation pattern in the *Car Rental* case study

4.4.4.4.2 Problem

The delegation of the *depender* actor element to the SSA can cause several changes in the entire organizational context; mainly, in the actors involved in the dependency relationship. These changes are related to the role played by the *dependee* actor, once the *depender* actor element is delegated. The influence of this delegation on the other organizational actors involved in the business must also be analyzed.

4.4.4.4.3 Forces

There are four forces associated to the solution of this pattern:

- The *depender* actor element is linked to a resource dependency (*dependum*).
- The *depender* actor element is linked to a plan dependency (*dependum*).
- The plans delegated to the SSA require the intervention of the original owner actors.
- The plans delegated to the SSA require the intervention of other organizational actors, not only of the original owner actors.

4.4.4.4.4 Structure

The elements used in this pattern are the following:

- **Organizational actors:** these are the *depender* actor as well as the *dependee* actor in the analyzed dependency relationship.
- ***Depender* actor element:** this is the element that needs to be delegated to the SSA. The element must be a plan or a goal.
- ***Dependum*:** this element represents the context around of the dependency; it must be a resource or a plan.
- **Dependency relationship:** this element joins the *depender/dependee* actors and the *dependum* object.

Scenarios:

There are three possible scenarios in which this pattern can be found in the organizational context:

Scenario I. This describes the situation where the *depender* element is a plan (it must be delegated to the SSA), and the *dependum* object is a resource. Therefore, the scenario represents the need of the business to obtain a resource in an automatic way. Figure 4.34 (a) depicts this situation.

Scenario II. This describes the situation where the *depender* element is a plan (it must be delegated to the SSA), and the *dependum* object is a plan. This scenario represents the automation of the *depender* plan, which delegates a plan to another actor. Figure 4.34 (b) depicts this situation.

Scenario III. This describes the situation where the *depender* element is a goal (it must be delegated to the SSA), and the *dependum* object is a plan. This scenario represents the delegation of a goal to the SSA which delegates the execution of a plan to another actor. Figure 4.34 (c) depicts this situation.

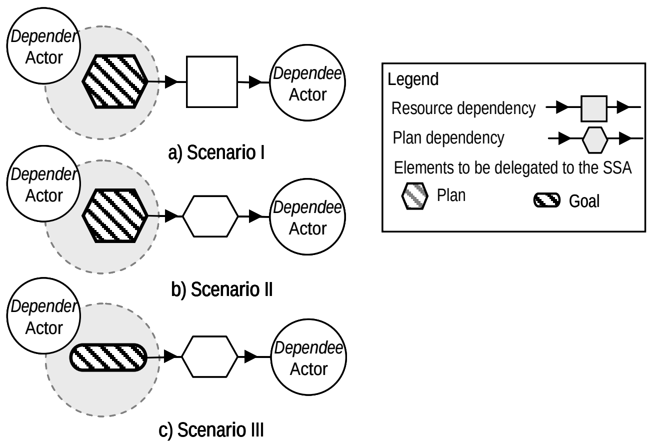


Figure 4.34 Scenarios of the depender element delegation pattern

4.4.4.4.5 Solution

The solution proposed for delegating only the *depender* actor element is guided by the *dependum* object. Therefore, when the object *dependum* is a resource, it will indicate the need to automate the reception of the resource. Otherwise, if the *dependum* is a plan, it will indicate the need for the execution of a plan by an organizational actor to fulfill the delegated plan or goal. This process is summarized in six steps:

Step 1. Delegate the *depender* actor element to the SSA.

Step 2. Analyze the *dependum* object in the dependency relationship under study; if the *dependum* is a resource, then the actor that will provide the resource to the SSA must be determined.

> **Step 2.1** If the *O-Dee* will play the role of *Provider* of information to execute the plan, (i.e., if the *O-Dee* provides the resource directly to the SSA) then the original resource

dependency is redefined between the SSA and *O-Dee* actor. The SSA will act as the *depender* actor. Figure 4.35 shows a scenario of the pattern described in this section (on the left). Thus, the element to be delegated is a plan, and the object *dependum* is a resource.

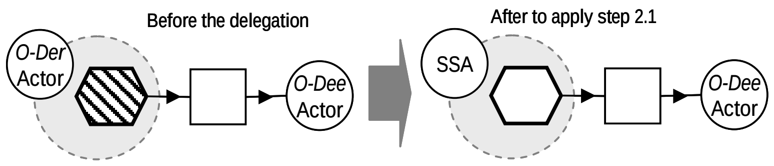


Figure 4.35 Organizational model before applying step 2.1 (the O-Dee actor acts as *provider* of information)

On the other hand, the model on the right shows the obtained solution before applying the step 2.1, where the resource dependency has been redefined between the SSA and the *O-Dee* actor to indicate that. *Dependee* actor will provide the resource to SSA directly.

Step 2.2 In contrast to step 2.1, if the *O-Der* is the actor that will play the role of *Provider* of information to execute the plan, then the original resource dependency remains the same, and another resource dependency must be created between the SSA and the *O-Der*. The *depender* actor of this new dependency will be the SSA. Figure 4.36 shows the alternative solutions for this substep. The *O-Der* actor acts as *Provider* of information, therefore the original resource dependency remains the same between the *O-Der* actor and the *O-Dee* actor, and a new resource dependency between the SSA and *O-Der* actor is created.

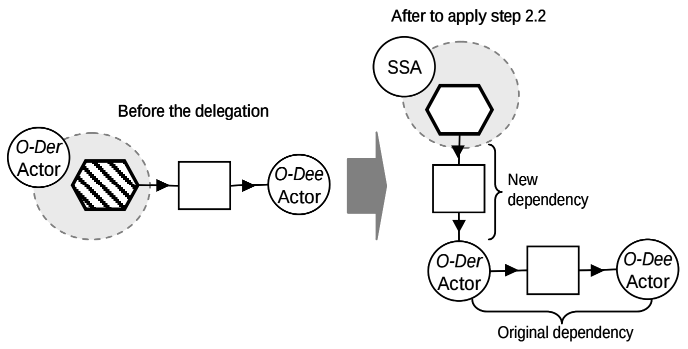


Figure 4.36 Organizational model after applying step 2.2 (the O-Der actor acts as *provider* of information)

> **Step 2.3** If no actor has any interaction with the SSA to execute the delegated plan, no dependencies must be created. The selection of this alternative implies the analysis of the plan name in order to make it appropriate for the new organizational configuration.

Step 3. If the *dependum* object is a plan, the organizational actor responsible to execute the plan dependency must be determined.

> **Step 3.1** If the *O-Der* is responsible for executing the plan dependency, the plan dependency must be redefined between the SSA and the *O-Der* actor. However, if the *O-Dee* actor is the one performing the plan of the plan dependency, then it must be redefined between the SSA and the *O-Dee* actor. Figure 4.37 shows the two scenarios where the object *dependum* is a plan. The first scenario shows a *depender* actor plan which must be delegated to the SSA; after applying step 3.1, the plan dependency must be redefined among some actors involved in the dependence (the *O-Der* or the *O-Dee* actor) and the SSA. The *depender* actor is the SSA. On the other hand, the second scenario of the figure shows a goal associated to a plan (Figure 4.35); after applying step 3.1, both organizational actors (the *dependee/depender)* can be responsible to execute the plan, in order to fulfill the delegated goal.

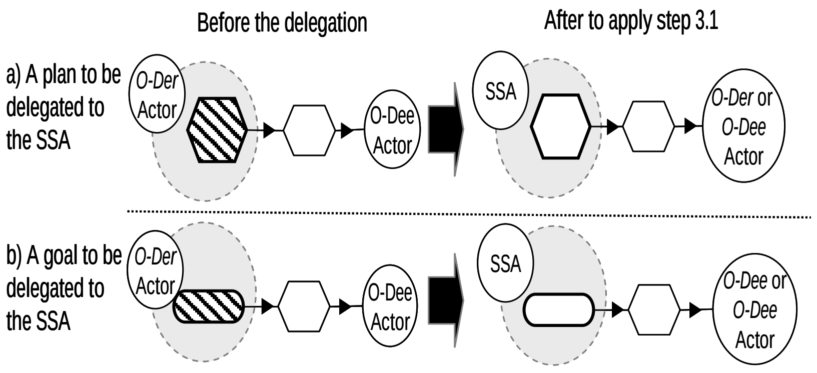


Figure 4.37 Two examples where the object *dependum* is a plan of the depender element delegation pattern

Step 4. Analyze the influence of the delegation of the *depender* actor plan on the organizational actors.

> **Step 4.1** When other organizational actors must provide information to the delegated plan, a new resource dependency between the actor and SSA must be created. The *depender* of this dependency will be the SSA.
>
> **Step 4.2** When other organizational actors need to obtain information about the delegated plan, then a new resource dependency between the actor and SSA must be created. The *dependee* actor will be the SSA.

Step 5. If more than one dependency relationship is generated during the delegation of the *depender* actor element to the SSA, they must be labeled with the same number in order to indicate their association.

4.4.4.4.6 Examples

The application of the steps of the *depender* element delegation pattern is illustrated with the example shown in Figure 4.33.

The first step in the solution of this pattern consists of delegating the plan *Obtain date and model car* to the *employee* actor to the *Car Rental* actor.

The second step is applied because the *dependum* object is a resource: *Info car*. Therefore, the different roles played by the actors must be analyzed. The roles and their solutions are grouped in three alternatives. They are specified as sub-steps. An example is shown for each alternative in the following paragraphs.
The first alternative solution for this pattern must be applied when the *Customer* (*O-Dee* actor) plays the role of *Provider* of information. Thus, the original resource dependency is redefined between the SSA and *Customer* actor. Figure 4.38 depicts this example.

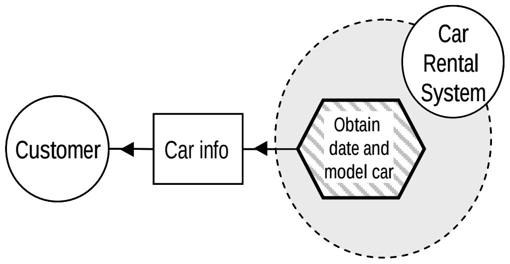


Figure 4.38 The O-Dee actor plays the role of *provider* of information

The second alternative solution for this pattern must be applied when the *O-Der* actor (*Employee* actor) plays the role of *Provider* of information to execute the plan. Thus, the original resource dependency remains the same, and another resource dependency is created between the SSA and the *Employee* actor. Figure 4.39 depicts this example.

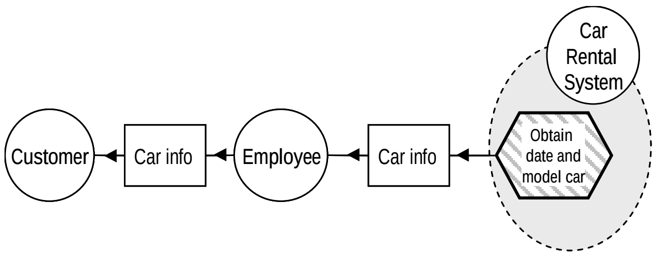


Figure 4.39 The O-Der actor plays the role of *provider* of information

The third alternative solution of this pattern cannot be applied in the delegated plan because it needs to obtain information about an organizational actor in order to be satisfied.
The fourth step is related to the creation of new dependency relationships among the organizational actors and the SSA in order to provide or require information about the plan delegated to the SSA. Finally, in the fifth step, the dependencies have been labeled with the number 1 to indicate their association

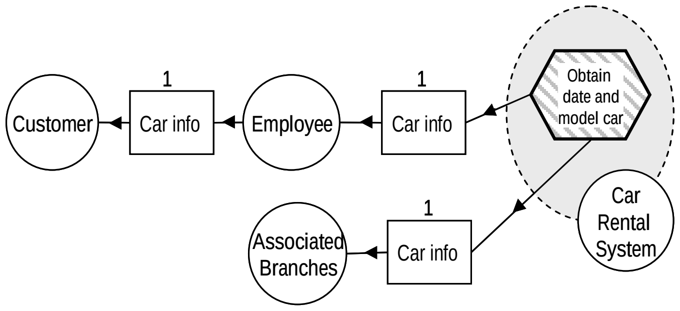


Figure 4.40 Organizational model after applying all steps of the pattern

4.4.4.5 The *dependee* element delegation pattern

This pattern must be used when only the element of the *dependee* actor needs to be delegated to the SSA to automate its execution; this element must be a plan, while that *dependum* object can be a resource or a plan.
The pattern details the problems that could be found in the delegation of the *dependee* element, and it shows the alternative solutions for that delegation. Figure 4.41 depicts an example of this structure.

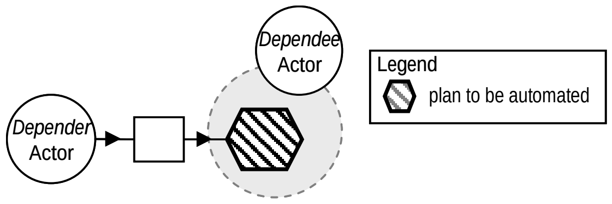


Figure 4.41 Example of the depender actor plan to be automated

4.4.4.5.1 Context

This pattern concerns the automation of the element of the *dependee* actor, which is associated by a dependency relationship. This pattern must fulfill the following conditions:

- One plan of the dependee actor joined by the dependency relationships needs to be delegated to the SSA
- The *dependum* object must be a resource or a plan

Figure 4.42 illustrates an example of this pattern in the *Car Rental* case study. Specifically, the figure represents the plan: *Manage the reservations*. This plan has been selected to be automated. For this reason; the plan needs to be delegated to the SSA.

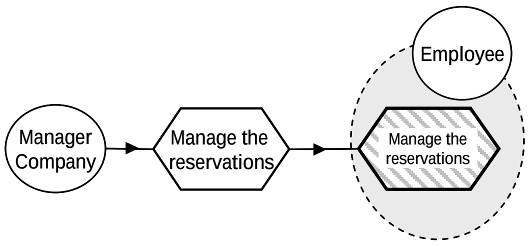


Figure 4.42 Example of the dependee element delegation pattern in the *Car Rental* case study

4.4.4.5.2 Problem

The delegation of the *dependee* actor element to the SSA can cause several changes in the entire organizational context; mainly, in the actors involved in the dependency relationship. These changes are related to the role played by the *depender* actor, once the *dependee* actor element is delegated. The influence of this delegation on the other organizational actors involved in the business must also be analyzed.

4.4.4.5.3 Forces

There are three forces associated to the solution of this pattern:

- The *dependee* actor element is linked to a resource dependency (*dependum*).
- The *depender* actor element is linked to a plan dependency (*dependum*).

- The plan delegated to the SSA requires the intervention of other organizational actors, not just the original owner actor.

4.4.4.5.4 Structure

The elements used in this pattern are the following:

- **Organizational actors:** these are the *depender* actor as well as the *dependee* actor in the analyzed dependency relationship.
- **D*ependee* actor element:** this is the element that needs to be delegated to the SSA. The element must be a plan.
- ***Dependum*:** this element represents the context around the dependency; it can be a resource or a plan.
- **Dependency relationship:** this element joins the *depender/dependee* actor and the object *dependum*.

Scenarios:

There are two possible scenarios in which this pattern can be found in the organizational context:

Scenario I. This describes the situation where only the *dependee* actor element must be delegated to the SSA; this element must be a plan, and the *dependum* object is a resource. Therefore, the scenario represents the need of the business to generate a resource in an automatic way through the *dependee* actor plan. Figure 4.43 (a) depicts this situation.

Scenario II. This describes the situation where only the *dependee* actor element must be delegated to the SSA; this element must be a plan, and the *dependum* object is a plan. This scenario represents the automation of the *dependee* plan, which was delegated by the *depender* actor. Figure 4.43 (b) depicts this situation.

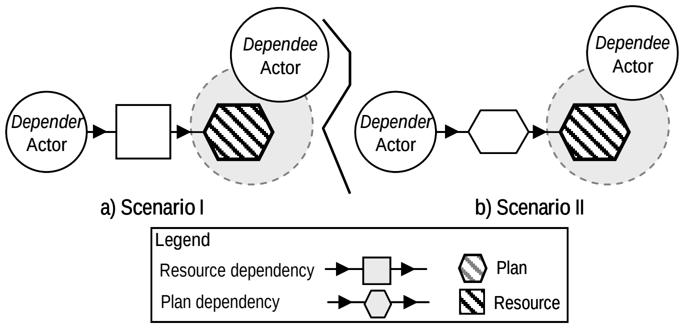


Figure 4.43 Scenarios of the dependee element delegation pattern

4.4.4.5.5 Solution

The solution proposed for delegating only the *dependee* actor element is guided by the *dependum* object. Therefore, when the object *dependum* is a resource, it will indicate the need to automate the generation of the resource. Otherwise, if the *dependum* is a plan, it will indicate the delegation of the *depender* actor plan to the *dependee* actor; this process is summarized in five steps:

Step 1. Delegate the *dependee* actor plan to the SSA.

Step 2. The *dependum* of the dependency relationship under study must be analyzed; if the *dependum* is a resource then the roles played by the organizational actors must be determined. It will be necessary to determine the actor that will provide the resource to the SSA.

> **Step 2.1** If the *O-Der* actor will play the role of *Requester* of information to execute the plan, (i.e., if the *O-Der* provides the resource directly to the SSA) then the original resource dependency is redefined between the *O-Der* actor and the SSA. The SSA will act as the *dependee* actor. Figure 4.44 shows a scenario of the pattern described in this section (on the left). Thus, the element to be delegated is a plan, and the object *dependum* is a resource. The model on the right of the figure shows the solution obtained after applying step 2.1, i.e., when the *O-Der* actor can access the SSA directly in order to obtain the generated resource by the delegated plan. Thus, the

original resource dependency is redirected from the *O-Der* actor to the SSA.

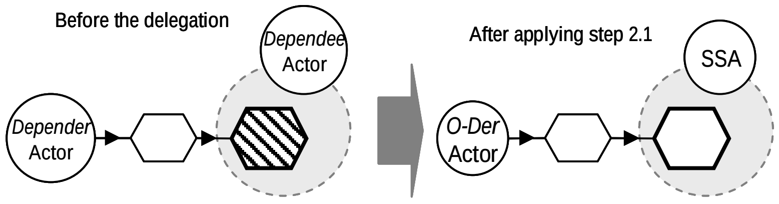


Figure 4.44 Organizational model after applying step 2.1 (the O-Der actor acts as *requester* of information)

Step 2.2 In contrast to step 2.1, if the *O-Der* does not have access to the SSA to obtain the resource generated by the delegated plan, the original resource dependency remains the same and another resource dependency must be created between the SSA and the *O-Dee*. The *dependee* actor of this new dependency will be the SSA.

Figure 4.45 shows the delegation of plan of the *dependee* actor. The plan has a resource dependency associated to it. Once the plan is delegated to the SSA, the dependency relationship remains the same between the organizational actors and a new resource dependency between the SSA and *O-Der* actor is created. In this case the SSA will act as *provider* of information of the delegated plan.

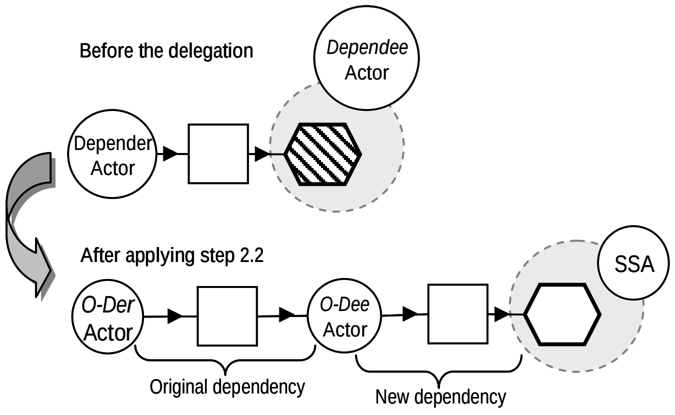


Figure 4.45 Organizational model after applying step 2.2

Step 3. Analyze the *dependum* object in the dependency relationship under study; if the *dependum* is a plan, the dependency plan must be redirected between the *O-Der* actor and the SSA. Figure 4.46 shows an example of this step.

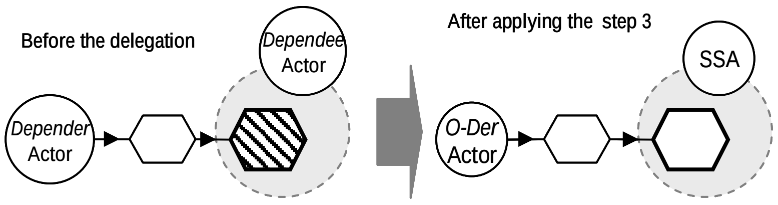


Figure 4.46 An example of the pattern when the *dependum* object is a plan

Step 4. Analyze the influence of this delegation in the organizational actors.

Step 4.1 When an organizational actor provides information to a delegated plan to execute it, a resource dependency between the actor and the SSA must be created. The *depender* actor of this new dependency will be the SSA. The new dependency indicates the reception of information from the organizational actor to the SSA.

Step 4.2 When an actor requires information from the delegated plan, a resource dependency between the actor and SSA must be created. The *depender* of this dependency will be the organizational actor. This new dependency indicates the delivery of information to the organizational actor from the SSA.

Step 5. If more than one dependency relationship is generated during the delegation of *dependee* actor plan to the SSA, then they must be labeled with the same number in order to indicate their association.

4.4.4.5.6 Examples

The application of the steps of the *dependee* element delegation pattern is illustrated with the example shown in Figure 4.42.

The first step in the solution of this pattern consists of delegating the plan: *Manage the reservations* to the *employee* actor to the *Car Rental System* actor. Second step must be omitted, because the *dependum* is not a resource. Next, step 3 is applied, and the plan dependency between the *Company Manager* actor and the *Employee* actor must be redirected. Figure 4.47 shows the delegation of the *employee* plan as well as the plan dependency redirected between the *Company Manager* and the *Car Rental System* system.

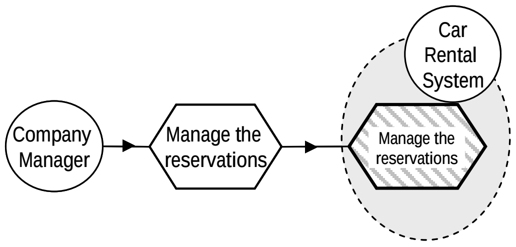


Figure 4.47 An example of the organizational model after applying step 3 of this pattern

The fourth step is related to the creation of new dependency relationships among the organizational actors and the SSA in order to provide or require information from the delegated plan to the SSA. Figure 4.48 continues with the example shown above. A new dependency is created between the *Employee* and the *Car Rental*

System; it indicates the need of the *Car Rental System* to obtain information about the company cars. Additionally in this picture the dependencies have been labeled with the number 2 to indicate the relation among them (fifth step).

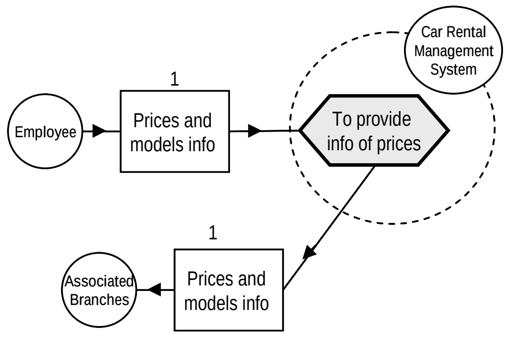


Figure 4.48 Organizational model after applying all the steps of the pattern.

4.5 Summary

One of the main problems of current research works on organizational modeling is the lack of a methodological approach to map the elements of an organizational model into the elements of a requirements model for the software system-to-be. Due to this methodological lack, efforts in the organizational modeling phase have not yet provided a practical solution for model transformation in software development environments.

In this work, we have proposed a pattern language which allows us to reduce the abstraction level of a “pure” organizational model so that it is closer to the requirements model. This process has been achieved by inserting the software system as an actor into the organizational model and redirecting the relevant tasks, goals and dependencies of the organizational actors to this new actor. In this way, there is a pattern for each situation that arises in the redirection of tasks or goals to the new organizational model. The new organizational model generated from the application of FELRE allows us to have a high-level description of the task that must be supported by the information system. This high-level description enables us to focus only on the relevant aspects to be automated, thereby reducing the complexity of the analysis task. The generated organizational model is, therefore, an intermediate model between the organizational model and the software requirements model. The proposed method complies with the MDA approach because it implements the concept of PIM-to-PIM transformations.

Figure 4.49 shows a partial view of the organizational model generated for the pattern language. This model includes the SSA and the actors that interact with it. The new organizational model represents a final result of the application of the goal analysis and the pattern language. In this model, the software system is represented as an actor (*Car Rental System*). The specification of the internal elements of this actor represents all the functionalities that this actor must provide for fulfilling the organizational goals. The model also represents the interactions among the organizational actors and the software system.

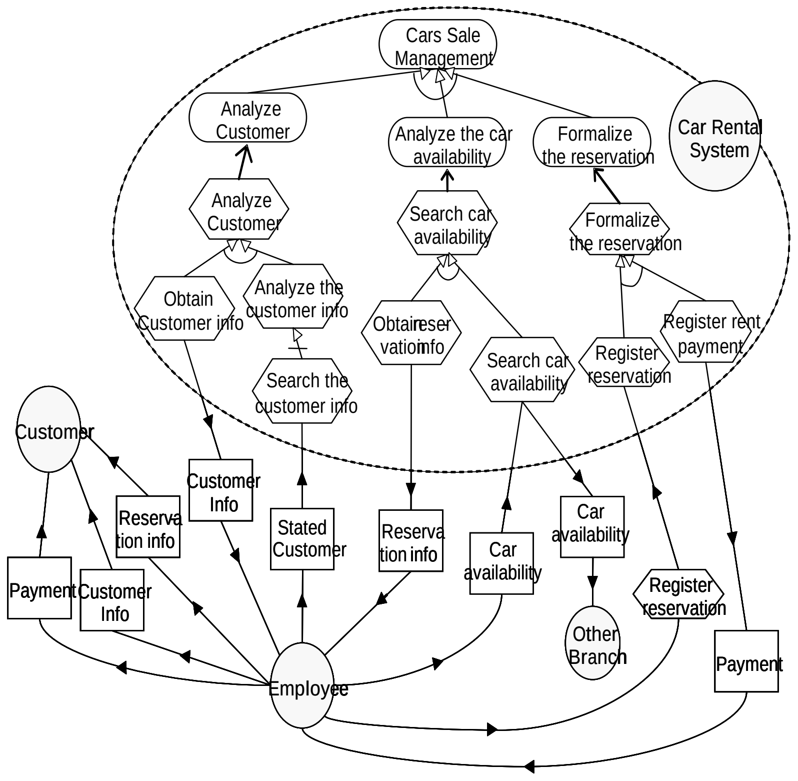


Figure 4.49 Partial view of the organizational model, which includes the software system actor

Part II

Late Requirements

Chapter 5

Extending the organizational models

In this Chapter, we introduce the process to insert monitoring plans into the organizational model. The monitoring plans enable the final user to supervise the business activities needed to satisfy the organizational objectives. Thus, we detail the generation process to extend the organizational model where the monitoring plans are defined. This model represents the relevant elements to be considered in the construction of the software system. The aim of this approach is to continue reducing the abstraction level between the early and late requirements models.

5.1 Introduction

At the present time, there is no definite solution to the problem of linking business (early) models with software (late) requirements models in a methodological way. One of the main reasons for the lack of solutions to this problem is the different nature of their specifications. In the early requirements phase, the concepts are related to the organizational context, whereas in the late requirements phase, the concepts are related to the software system to be developed. There is a significant difference between the abstraction levels of the two requirements specifications.

The proposed method allows us to carry out a soft transition between early and late requirements phases by detailing those elements that are relevant in the construction of a software system. It is important to point out that some of these elements may not be considered as relevant when trying to understand the business context because they are only important in defining its automation through an information system.

Another contribution of this thesis is the insertion of monitoring plans in the organizational model. The aim of monitoring plans is to prevent or detect undesired behaviors in the system-to-be in order to take the corrective measures to manage them.

5.2 The late requirements phase

The late requirements phase is mainly focused on describing the system to-be within its operational environment along with the relevant functions and qualities [Cast02].

We propose to extend the organizational model by representing the objects of interest (concerned objects) associated to the relevant plans and resources to be automated. This new model, which contains the concerned objects, allows us to analyze the flow of information in the enterprise and also permits us to analyze the lifetime of the information managed in the organizational processes.

The organizational model that is extended with the concerned objects will be the basis for the systematic generation of a requirements

model, which is represented by using the use case models. The extended organizational model is also the basis for the generation of a conceptual model that is compliant with OO-Method.
Figure 5.1 shows an overview of the processes that make up the late requirements phase. The inputs in this schema are the plans to be automated that were identified using the goal-based requirements elicitation process. The deliverables of these processes are: the organizational model that is extended with the concerned objects and the scenarios of the concerned objects.

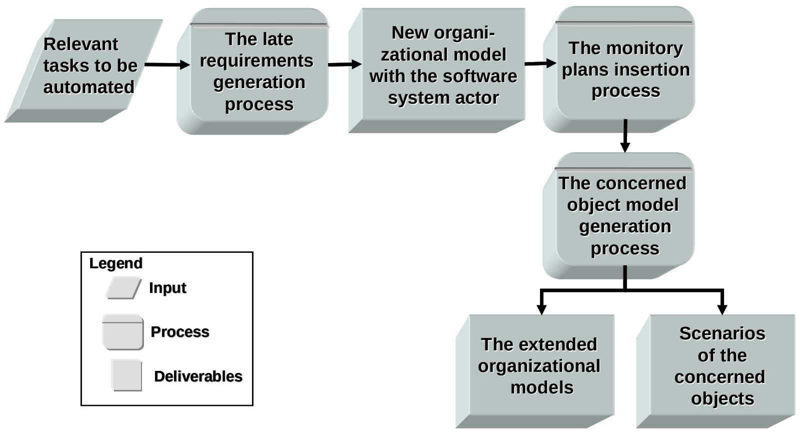

Relevant tasks to be automated
The late requirements generation process
New organi-zational model with the software system actor
The monitory plans insertion process
The concerned object model generation process
The extended organizational models
Scenarios of the concerned objects
Legend
Input
Process
Deliverables

Figure 5.1 Processes of the late requirements phase

A set of rules and algorithms that allow us to systematically carry out the transformations between models are presented. Then, the concept of monitoring is detailed.

5.3 What is Monitoring?

Monitoring is often used to make a multi-agent system more robust in the presence of undesirable behaviors such as faults. Several approaches address the problem of monitoring in multi-agent systems. They rely on events and their goal is to observe, analyze and control the behavior of the system. These approaches usually observe the execution of the multi-agent system in order to define its current behavior model and correct the undesirable behaviors.

There are several research works about the difficulties of using monitoring to control undesirable situations.
Castelfranchi [Cast98], and Sichman and Conte [Sich02] introduce interdependence graphs that are used to predict some undesirable situations (e.g., inequity or incompatibility). Their analysis relies on knowledge that is defined a priori, such as the number of agents, their plans, their goals, and their relations of interdependence.
Kaminka et al. [Kami02] propose a monitoring approach in order to detect and recover faults. This approach uses models of relations between mental states of agents. These authors adopt a procedural plan-recognition based approach to identify inconsistencies. They argue that any failure comes from incompleteness of beliefs.
The works of Horling et al. [Horl01] present a distributed system of diagnosis. The faults can directly or indirectly be observed in the form of symptoms by using a fault model. The diagnosis process modifies the relations between tasks, in order to avoid inefficiencies.
There are also monitoring techniques that are mainly used in the analysis of problems presented in dynamic environments [Feat98] [Kous04] [Fick95] [Cohe97] [Gues04].

5.4 The monitoring plans insertion process

The addition of monitoring plans in the organizational model is one of the contributions of this thesis. The main advantage of the proposed technique is that it provides ongoing verification of progress toward achievement of objectives and goals. The inclusion of monitoring plans permits the business activities to be supervised, observed, and tested and appropriately reported to the responsible actors.
Our research work is focused on the analysis of the system to be to define monitoring plans that must be installed to gather and analyze pertinent information about the system’s run-time environment. Therefore, we must detect those situations that adversely affect the execution of the organizational processes. We need to analyze the organizational context in which the system will be implemented,

how the organizational elements (goals, resources, planes) can be affected, and who can help to solve these situations.
The monitoring analysis is applied in the organizational model that has been extended by including the software system as a organizational actor. In this work, the monitoring is implemented by using preconditions that monitor the organizational plans. We consider that the result of the monitoring process can play a relevant role in determining the elements to be considered in the construction of the system-to-be.
Nevertheless, the monitoring plans insertion process detailed here requires great experience from the requirements analysts. This is because the analyst must identify the elements and information that can be affected during the system run-time. We propose some guidelines to insert the monitoring plans into the organizational model. The aim of these guidelines is to identify all the factors that can affect the construction of the software system.

5.4.1 Monitoring plans and monitoring data

According to the Tropos approach, a plan represents a way of doing something at an abstract level. When the plans are contained inside the software system actor, the plans represent the actions that must be supported by the system-to-be. The monitoring plans need to be defined once the relevant plans to be automated through the software system have been identified and delegated to the system actor. Following, we detail the steps to carry out the insertion of monitoring plans into the organizational model.
Step 1: Identify the plans to be monitored.
The first step consists of identifying the critical plans to be monitored. These plans will be selected from the organizational model which includes the software system actor.
In the *Car Rental* case study, the plan *Register reservation* of the software system actor contains the reservation data (where a car is assigned to a reservation) that is generated when the reservation is registered in the software system. However, there are several situations where the assigned car may not be available at the time of delivery. Thus, the plan *Register reservation* is a candidate to be a

monitored plan. The analyst must determine all the plans that could be affected by unexpected situations. These plans are candidates to be considered as monitoring plans.

Step 2: Determine the parameters to be monitored.

This step is carried out after the monitoring plans have been identified. Each executed plan in the organizational model can be associated to one or various resources that are created or modified during the execution plan. Therefore, although the monitoring plans are the central topic at this stage, it is also necessary to identify what resources must be monitored.

Once the resource candidates to be monitored have been identified, the conditions that activate the monitoring must be determined. For example, the plan *Register reservation* has several associated resources: car, reservation date, etc. In this case, *car* is the resource that must be monitored. The condition that must be monitored is the availability of the car that has been booked.

Step 3: Insert dependencies and new organizational plans.

In this step, new dependencies and plans must be inserted in the organizational model in order to represent the actions (monitoring body) that must be taken when the conditions for the monitoring plans are reached.

The elements that compose the monitoring body will the be plans executed by the SSA or by other organizational actors. In the first case, the monitoring plans are defined as internal activities in the software system actor. In the second case, the plans are represented as strategic dependencies between the SSA and other organizational actors. Figure 5.2 shows both schemas, where the preconditions to be satisfied are represented as new plans to be executed inside the software system actor.

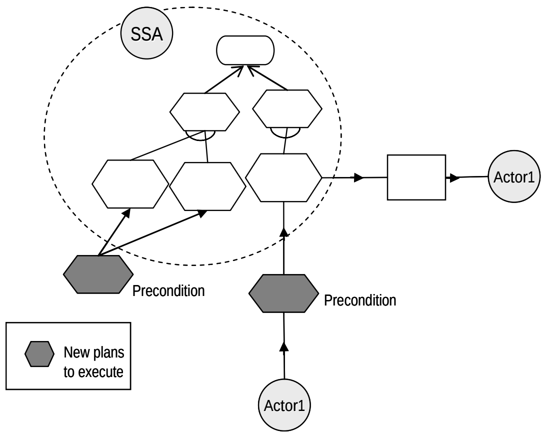


Figure 5.2 Schema of the monitoring plans in the organizational model

Step 4: Determining the success conditions.
This step consists of determining the success condition. By success conditions, we mean the conditions that eliminate the need to monitor a plan, so there is no reason to monitor it anymore. For example, when a car is delivered to a *Customer*, the plans that monitor the car availability must be terminated.
We propose four dimensions to characterize monitoring plans. Table 5.11 shows these four dimensions. The first column, ***Monitoring plan***, contains the name of the plan to be monitored. The second column, ***Parameters of the monitoring***, contains the condition to be monitored. The third column, ***Period of time to carry out the monitoring***, contains the period of time which an element must be controlled. Finally, the fourth column, ***Activities to do***, describes all the activities that must be executed if monitoring parameter is affected.

Table 5.1 Dimensions to characterize monitoring plans

Monitoring plan	Parameters of the monitoring	Period of time to carry out the monitoring	Activities to do

5.5 Extending the organizational models with the identification of concerned objects

This section describes the process to extend the organizational model in order to identify the relevant information[1] in the construction of the system-to-be. There are two main sources for this information: a) the plans performed by the organizational actors including the SSA and b) the characteristics of the resources used, modified or generated as a result of the execution of the organizational plans, in the organizational context. The process to extend the organizational model with the concerned objects is detailed by using rules, steps and algorithms that enable the analyst to generate a new extended organizational model. The following section describes the concepts, notations and the models that are generated in the process of extension of the organizational model.

5.5.1 Concepts and models

This subsection presents our definition of concerned object, which is a key element in the transformation process of early to late requirements. We also detail both models that are generated: the extended actor diagram and the extended goal diagram. Finally, the scenarios of the concerned objects are also detailed.

Definition of a concerned object

The term *Concern* has been widely used in the literature associated to software engineering. It is often found in techniques that focus on aspect-oriented software development (AOSD) [Sutt04].

A concern expresses a specific interest in some topic pertaining to a particular system of interest (or other subject matter) [Hill99]. It is important to point out that concerns do not exist until someone is *concerned* about them. For example, in our proposed method, a plan

[1] By *relevant information,* we mean the information needed in the construction of an information system.

does not constitute a concern until an analyst has some reason to be interested in a plan as candidate for functionality in the system-to-be.
We use the concept of *concerned object* to represent an entity of interest in the process of defining the system-to-be. Therefore, a concerned object represents a resource that is used within the organizational process or an abstract entity of information that will be used in the software system-to-be.
A concerned object is represented by a circle, and its attributes are represented by small circles that are associated to the concerned object. The name of the concerned object and the names of the attributes must be placed within each circle. Figure 5.3 shows an example of a concerned object with its set of attributes.

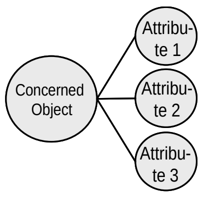


Figure 5.3 Primitives of the concerned object model

The concerned object model

The concerned object model is an extension of the actor diagram and the goal diagram of the Tropos framework. The proposed extensions focus on describing the relevant information in the software system-to-be through of the identification of concerned objects in the elements of each organizational model.
The following subsections describe the structure of the extended actor and goal diagrams. The structure of the scenarios of the concerned object is also explained.

The extended actor diagram

The actor diagram is made up of the organizational actors, who are related to other actors through dependency relationships. Therefore, the main sources for the detection of the concerned objects are the resource and plan dependency relationships, which involve organizational plans to send and receive information to/from the software system to-be. The extension of the actor diagram involves the representations of the concerned objects associated to the resource and plan dependencies.

Figure 5.4 illustrates an actor diagram that is composed of two organizational actors and the SSA. There are two resource dependencies between the actors that show the flow of information between them. Therefore, the extension of this diagram is carried out over these dependencies.

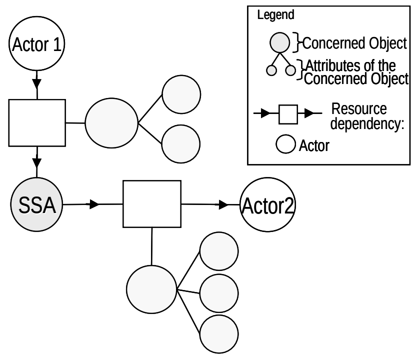


Figure 5.4 The concerned object model (actor diagram)

The extended goal diagram

The goal diagram which is focused on actor activity provides a microscopic view of the application domain. Therefore, the extension of this diagram is related to the internal elements of each actor (more specifically to actor plans and goals). The plans that contain the SSA always involve manipulation of informational entities that are relevant for the system-to-be. Thus, the plans in the SSA must be extended with their corresponding concerned objects. However, not all the goals need to be analyzed; only those that are related to a set of plans by means-end links need to be extended because these goals involve informational entities through plans.

Therefore, the internal plans in the goal diagram can be extended by one or several concerned objects, depending on the information or resources used in the execution of the plan. For example, a parent plan can be composed of the set of concerned objects that are identified in its child plans. In this way, the internal goals that

represent an *"end"* in a *means-end* will be extended with the concerned objects that are identified in their child nodes.
Figure 5.5 shows two actors in a goal diagram. The elements of the SSA have been extended with the identification of concerned objects. For example, the SSA has a goal (Goal$_1$) associated to three plans (Plan$_1$, Plan$_2$, Plan$_3$) by a *means-end* link. Plan$_1$ is extended by the concerned object *"A"*. Plan$_3$ is also extended by the same concerned object.

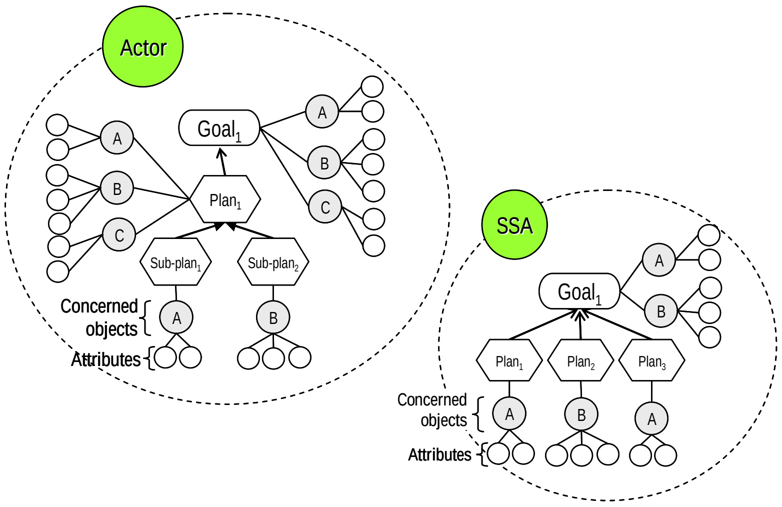


Figure 5.5 Concerned object model (goal diagram)

This situation represents the use of the same resource or information in different plans executed in the organizational context. Plan$_2$ is extended by the concerned object "*B*". Thus, the goal G$_1$ of the SSA is extended with the concerned objects of its associated plans (concerned objects: *"A"* and "*B*").
The extension of the elements of the organizational actor is carried out in the same way as in SSA. However, in this actor, plan$_1$ identifies the concerned object "*A*" and "*B*" from its *subplans*, and also identifies the concerned object *"C"*.

Defining scenarios for each concerned object

As a natural consequence of sharing information in the organizational context, the concerned objects identified in the actor and goal diagrams can be identified in more that one element of the diagrams. More specifically, the detected concerned actors can have different attributes depending on the element where the concerned object is identified. For example, the concerned object *"A"*, which has been identified in the SSA of Figure 5.5, could contain different attributes each time that the concerned object is identified in the organizational model. Therefore, when a concerned object is identified, the set of characteristics must be stored in order to obtain the global characteristics of each concerned object. A scenario is a situation where a concerned object is identified. The aim of the proposed method is to capture the scenarios of each concerned object.

The information of the analysis of concerned actors is stored in a table; each column represents a parameter of the scenario and each row represents the information of the concerned object each time that it is used. Table 5.2 depicts the scenarios of the concerned objects. Each parameter is explained in detail. The first column, ***Concerned object name,*** contains the name of the identified concerned object; the second column, ***Elements and associated links,*** contains the type of element where the concerned object was identified (for instance, a plan or a resource dependency or a *subplan*, etc.); the third column, ***Associated elements,*** is only used for the concerned object identified in the goal diagram. If the element that leads the concerned object is involved in a *means-end* or *and-or* relationship with other internal elements, then the value of this column indicates the name of the parent node of the element that leads the concerned object. The fourth column ***Used attributes,*** contains the attributes used in the concerned object; the fifth column, ***Related actors,*** contains the actors of the dependency (when the concerned object has been located in the actor diagram) or the name of the actor where the concerned object was identified (when the concerned object is located in the goal diagram); finally, the sixth column, ***Label of the concerned object***, contains the state of the concerned object.

For example, when a concerned object is analyzed (to determine its space of alternatives) then the label "*Analyzed*" is placed in the object scenario. In another example, when the attributes of a concerned object are divided in order to generate other concerned objects the label "*divide*" must be placed in this column.

Table 5.2 Scenario of concerned objects

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object

5.6 The generation process of the concerned objects model

This section describes the generation process of the concerned objects that is proposed in this thesis. Figure 5.6 shows the schema of the generation process of the concerned objects model. The inputs for this process are the organizational models (actor diagram and goal diagrams) where the SSA is included. The process is summarized in two stages: 1) the first stage consists in the identification of the concerned objects and scenarios; 2) the second stage consists in the reconciliation of the various scenarios for the same concerned object. Finally, the outputs of the process are the extended organizational models and the scenarios of the identified concerned objects.

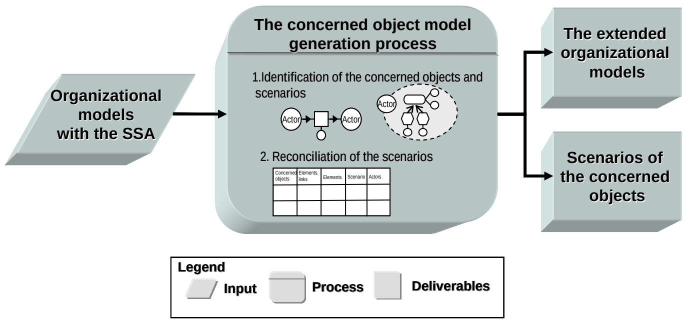

Organizational models with the SSA
The concerned object model generation process
1.Identification of the concerned objects and scenarios
Actor
Actor
Actor
2. Reconciliation of the scenarios
Concerned objects
Elements, links
Elements
Scenario
Actors
The extended organizational models
Scenarios of the concerned objects
Legend
Input
Process
Deliverables

Figure 5.6 The schema of the generation process of the concerned objects model

5.6.1.1 Identification of the concerned objects

As stated above, the concerned objects can be identified in the resources, plans and goals, of the organizational model. Therefore, attributes of a resource or the attributes used by a plan or goal must be analyzed. It is important to point out that resources can be found in dependency relationships or in parameters of a organizational plan. However, goals and plans need to be analyzed as internal elements in an actor or where they are represented as dependency relationships.

Algorithms for the generation process of the concerned objects model

The generation process of the concerned objects model is guided by two algorithms. Figure 5.7 shows the first algorithm which details the extension of the actor diagram. Figure 5.8 shows the algorithm to extend the goal diagram. Both of these diagrams include the SSA. These figures provide an overview of the algorithms of the generation process of the concerned objects model, which is represented by a control flow chart. The boxes in the figures represent the activities that an analyst must perform to extend the diagrams; these activities include the activation of a function or the application of a rule. The diamonds represent the various inquiry points. Diamonds have two exit points: one exit represents an answer of `*yes*´ and another exit indicate an answer of `*no*´. The arrows denote the flow of the process as well as the iterations in the process. The ellipses represent the inputs of the process and the outputs of the process.
The set of rules and functions indicated in the proposed algorithms are explained below in detail.

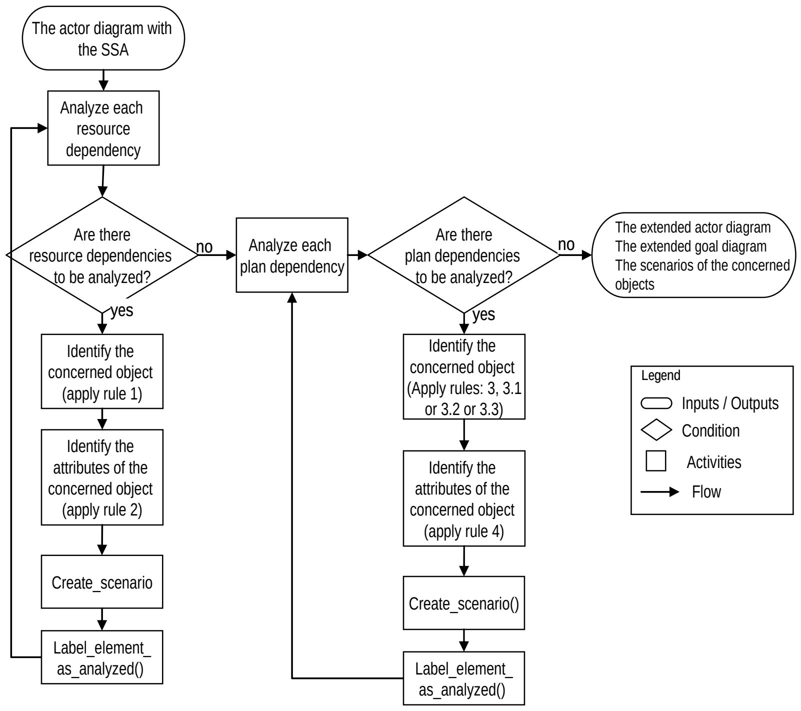

The actor diagram with the SSA
Analyze each resource dependency
Are there resource dependencies to be analyzed?
no
yes
Identify the concerned object (apply rule 1)
Identify the attributes of the concerned object (apply rule 2)
Create_scenario
Label_element_ as_analyzed()
Analyze each plan dependency
Are there plan dependencies to be analyzed?
no
yes
Identify the concerned object (Apply rules: 3, 3.1 or 3.2 or 3.3)
Identify the attributes of the concerned object (apply rule 4)
Create_scenario()
Label_element_ as_analyzed()
The extended actor diagram
The extended goal diagram
The scenarios of the concerned objects
Legend
Inputs / Outputs
Condition
Activities
Flow

Figure 5.7 The extended actor diagram control flow chart

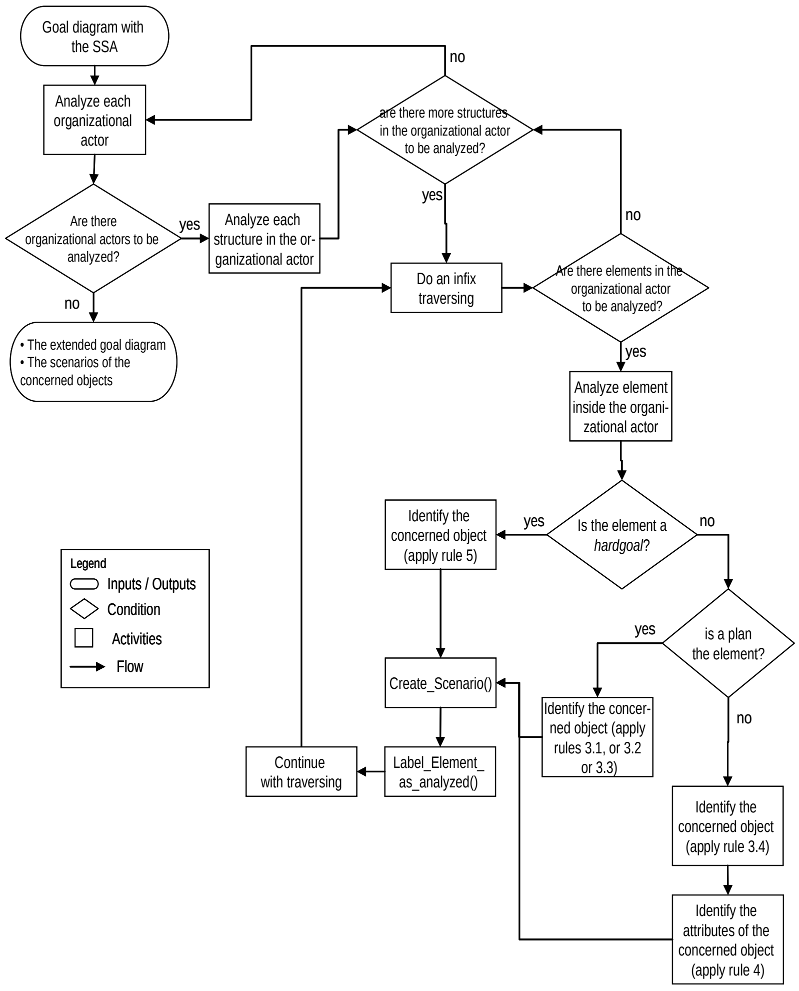

Goal diagram with the SSA
Analyze each organizational actor
Are there organizational actors to be analyzed?
yes
no
Analyze each structure in the organizational actor
are there more structures in the organizational actor to be analyzed?
no
yes
Do an infix traversing
Are there elements in the organizational actor to be analyzed?
no
yes
Analyze element inside the organizational actor
• The extended goal diagram
• The scenarios of the concerned objects
Is the element a *hardgoal*?
yes
no
Identify the concerned object (apply rule 5)
is a plan the element?
yes
no
Identify the concerned object (apply rules 3.1, or 3.2 or 3.3)
Identify the concerned object (apply rule 3.4)
Identify the attributes of the concerned object (apply rule 4)
Create_Scenario()
Label_Element_as_analyzed()
Continue with traversing
Legend
Inputs / Outputs
Condition
Activities
Flow

Figure 5.8 Extended goal diagram control flow chart

Rules for identifying concerned objects

Rule 1: A resource dependency between the SSA and another organizational actor can be extended with one or several concerned objects.

This is determined by the attributes of the resource. Therefore, the attributes with similar semantics must be grouped together to generate a concerned object. The following consideration must also be taken into account: When the attributes of the resource refer to the information of a organizational actor, the name of the concerned object must be the same as the analyzed actor.

Rule 2: The attributes of the resource will be the attributes of the created concerned object. The attributes of each identified concerned object will be assigned according to the groups created in rule 1.

Example of rules 1 and 2: Figure 5.9 shows a resource dependency, *Customer data*, which contains various parameters (Name, Passport-number, Address, City, Home-phone, License, and Birthday). These parameters and the name of the resource generate the concerned object *Customer*. This concerned object is created because all the information of the resource describes this actor.

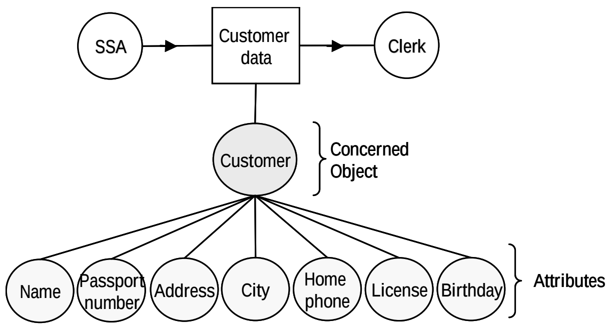


Figure 5.9 Example for extending a resource dependency

Rule 3: An organizational plan executed in the organizational context can be extended with one or more concerned objects. This will be determined by the resources that are used or modified during plan execution. The type of relationships or links associated to the plan must also be taken into account in the identification of the concerned objects. The following rules detail the process of extension of a organizational plan:

Rule 3.1 When a plan uses or modifies a resource, the plan must be extended using this resource, which is represented as a concerned object.

Rule 3.2 If a plan uses or modifies a resource that has not yet been identified as a concerned object, then the plan must be extended using this resource to create a concerned object.

Rule 3.3 When a plan does not use or modify any resource, then the plan does not need to be extended with a concerned object.

Rule 3.4 A *composite* plan[1] needs to be extended with the concerned objects that include its children nodes. For example, Figure 5.10 shows the structure of a *composite* plan and its associated *subplans*. Thus, the concerned objects identified in the *subplans* are used to define the concerned objects of the *composite* plan.

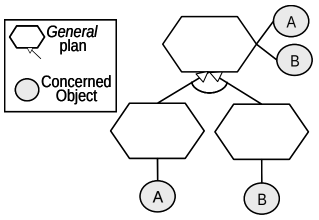


Figure 5.10 Example for extending a *composite* plan

Rule 4: The characteristics of the resources used in the performance of a plan must be used in the identification of the associated attributes of the concerned object identified.

Rule 5. A *hardgoal* is a candidate to be extended with concerned objects if the goal is involved in a *means-end link* where the children nodes are plans (Figure 5.11).

[1] This type of plan was defined in Chapter 4.

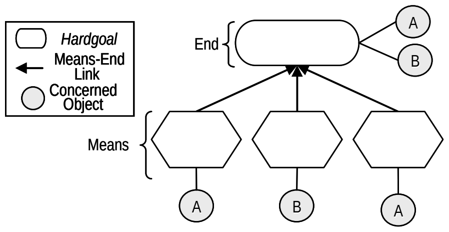


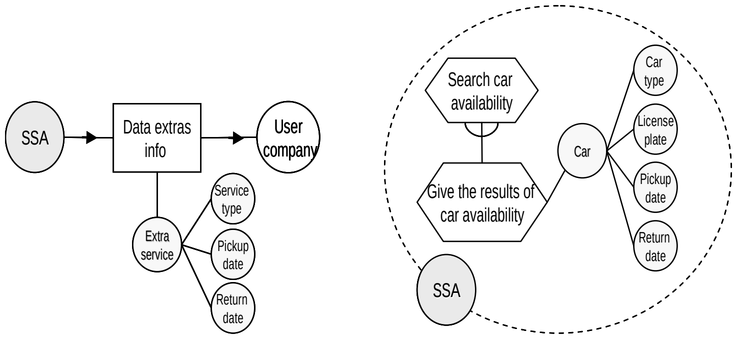
Figure 5.11 Example for extending a goal in a means-end link

Functions

The functions used in the algorithm are the following: Created_Scenario() and Label_element_as_analyzed().

Create_Scenario()

The main objective of this function is to store the different scenarios for each identified concerned object. The output of this function is the set of scenarios of each concerned object in the organizational model. For example, the scenario created for the concerned object *extra service* that is identified in a resource dependency (Figure 5.12, (a)) must contain the information shown in the first row of Table 5.3. This row indicates the following: a) the name of the concerned object is "*Extra service*" (first column); b) the concerned object has been identified in a resource dependency (second column); c) The value for the third column is empty for this example because the analyzed element correspond with a dependency relationship and not an internal element; d) The attributes used by this resource (*Extra services type, Pickup date, Return date)* will be used to create the attributes of the concerned object (fourth column); e) the name of the actors involved in the analyzed dependency: *SSA (Car Rental System)* and *User Company* (fifth column); f) the last column will be used in later processes.



(a) Extension in a resource dependency (b) Extension in a subplan

Figure 5.12 Example of two concerned objects

Another example of a concerned object in the goal diagram is shown in Figure 5.12, (b). The concerned object *Car* is shown with all the attributes that are used in this state. This concerned object is also located in a subplan (*Give the results of car availability*) that is linked to its parent node by an *AND* decomposition link (*Search car availability*). Thus, all this information must be placed in Table 5.3.

Table 5.3 Table of scenario concerned objects

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
Extra service	Resource Dependency (Data extras Info)		Extra service type, Pickup date, Return date	SSA-User Company	
Car	Sub-plan AND decomposition (Obtain extra service data)	*Composite* Plan (Search car availability)	Car type, Plate, Pickup date, Return date	SSA	

Label_element_as_analyzed()

The main objective of this function is to label each organizational element in the diagram in order to know which elements have been analyzed (the sixth column of the table of scenarios).

5.7 Summary

This Chapter has presented two relevant processes in mapping early and late requirements phases: the monitoring plans insertion process and the generation process of the concerned objects model.

The first process permits the definition of organizational plans to supervise and observe the activities of the business. We have proposed a set of steps for carrying out this process; however, despite the guidelines presented in this thesis, this process of definition of monitoring plans requires great experience on the parts of requirements engineers. This is because an in-depth analysis is necessary to identify the plans and the conditions that must be monitored.

The second process extends organizational models in order to identify the relevant information in the construction of the software system.

The chapter 8 shows a partial view of the goal diagram for the *Car Rental* case study with the concerned objects (Figure 8.12).

Chapter 6

Linking late requirements with the *ONME* conceptual model

A methodological approach to generate the *ONME*[1] object-oriented conceptual models from late requirements specifications is presented in this Chapter. The proposed approach provides a number of steps that enable the analyst to systematically use an organizational model to specify an alternative set of conceptual models that reflect different possibilities to represent the static structure of the system to-be constructed. From this alternative solutions space, a specific one is selected to be the input of the *ONME* CASE tool that makes the automatic correspondence among the conceptual schema and the

[1] Olivanova Model Execution

software system. This Chapter also introduces an overview of the *ONME*, the software generation method used in this research work.

6.1 Introduction

Software development is an activity that is increasingly complex and that requires powerful techniques of elicitation, specification and development. Software engineering has proposed several techniques, methodologies and tools to achieve the goal of developing information systems that appropriately fit user needs.

At the present time, there are very few research studies that are focus on the problem of the mapping process between the elements of a organizational model and the elements of a conceptual model that reflects the structure of the system to be developed [Alen00] [Jian06] Sant02]. We argue that this is a key phase in the software development process because there is a strong relationship between the semantics of the organizational models and the expected behavior of the system-to-be.

We also argue that the process of mapping both organizational and system specifications needs to be carried out in a systematic manner in order to ensure its application in real software development environments.

A methodological approach is proposed in this thesis to guide the generation of an object-oriented conceptual model for the system-to-be from a organizational model specification according to a goal-oriented approach. Design issues are used to select the relevant plans to be automated, which are represented as plans in the software system actor. A pattern language is proposed to carry out this process in a systematic manner. Then, design issues are addressed to select the appropriate conceptual model. To fulfill the objectives of this thesis, we have merged two well-known techniques: The Tropos Framework to represent the organizational model with a social an intentional approach; and the *ONME* software generation method to represent the conceptual model and also to provide automatic software generation from the conceptual schema. The proposed approach is illustrated step by step by using the *Car Rental* case study as running example.

6.2 Some considerations about the *ONME* Conceptual Model

In traditional requirements approaches, the conceptual model represents "the semantics of the actual data in the proposed database; its design focuses on issues that are specific to the conceptual content and organization of the data" [Jian06]. In this approach, an object-oriented conceptual schema is considered to be the resultant model of the requirements analysis, where the schema represents the model of the database. In this context, a UML object-oriented class diagram is equivalent to an ER[1] diagram as a database design technique. In this approach, the conceptual model is used to define the logical design of the database through, for example, a relational database schema in SQL.

However, in the *ONME* method, a class diagram represents more than the database schema. The *ONME* conceptual model represents not only the database, but also the structure of the software system.

In this context, we can establish some differences among class diagrams as system generators and class diagrams as database schemas.

- "Conceptually, an object does not need a key or other mechanism to identify itself, and such mechanisms should not be included in models." [UML07]. However, keys are very relevant in DataBase Design.
- A class is similar to an entity type, only operations are added.
- Classes do not necessarily describe persistent objects that might ultimately be stored as rows in a relational table.
- Classes can describe transient objects that exist only for the duration of the program execution.

[1] Entity Relationship

- Abstract classes only describe fundamental characteristics of low-level classes; therefore no examples can be defined for abstract classes.

The key differences among class diagrams are operations and keys. However, in this thesis, the main difference is that class diagrams can model the database and the database application system, so class diagrams represent not only the database design, but also the software system-to-be.

6.3 The *ONME* conceptual model generation

The *ONME* [Past96] [Past97] is an object-oriented software production method (Figure 6.1). Basically, we can distinguish two components: the Conceptual Model and the Execution Model. When facing the conceptual modeling step of a given information system, we have to determine the components of the object society without being worried about any implementation considerations. In the Problem Space level a precise system definition is obtained by means of a conceptual model.

In this context, we specify three models: the Object Model, the Dynamic Model and the Functional Model. They describe the object society from three complementary points of view within a well-defined OO framework. Then, we specify the Presentation Model using interface patterns in order to collect the interface information required to generate an interface that is ready to be used in an automated way.

Once we have an appropriate system description, a well-defined execution model (in the Solution Space level) will fix the characteristics of the final software product, in terms of user interface, access control, method activation, etc. According to the execution model a prototype that is functionally equivalent to the specification is built in an automated way. The code generation strategy is independent of any concrete target development environment; even if at the moment our selected environments for automated code generation are Visual Basic, Java, ColdFusion and JSP.

Next, we give a short overview of the four models (object, dynamic, functional and presentation) that constitute the conceptual model.

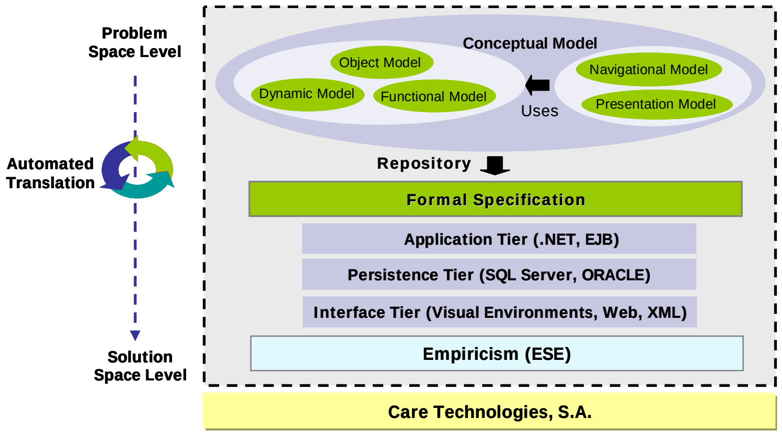


Figure 6.1 The *ONME* conceptual model generation approach

6.3.1 The *ONME* conceptual model

The *ONME* [Past01] [Past98] is a Model-Driven Development (MDD) approach [Seli03] which automatically generates *complete* object-oriented systems based on the information contained in the conceptual models.

The key feature of this proposal is the integration of formal specification techniques with conventional object-oriented modeling techniques. The main advantage is that the models are built using concepts that are much closer to the problem space domain. In addition, this integration avoids the complexity associated to the use of formal methods.

In a MDD approach, two main aspects must be clearly stated: which *conceptual modeling patterns* are provided by the method and which *notation* is provided to properly capture those conceptual modeling patterns. Figure 6.2 shows an abstraction of the MDD approach provided by the *ONME*.

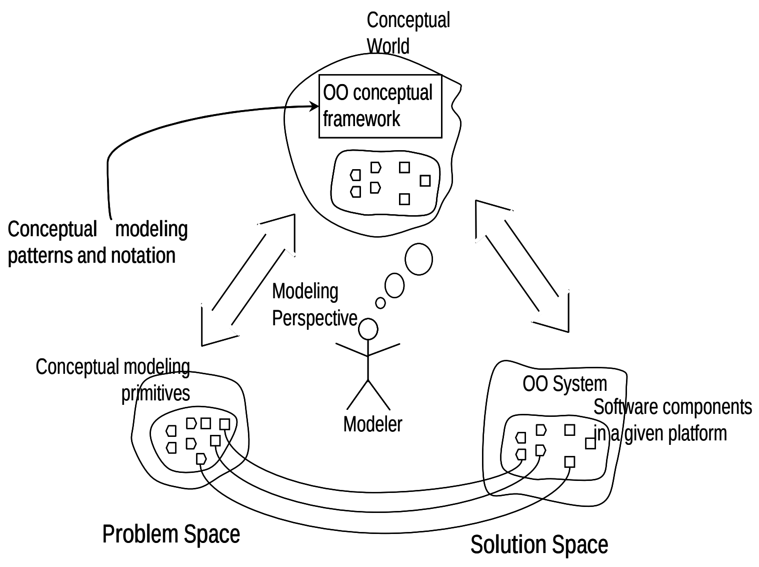

Conceptual World
OO conceptual framework
Conceptual modeling patterns and notation
Modeling Perspective
Conceptual modeling primitives
Modeler
OO System
Software components in a given platform
Problem Space
Solution Space

Figure 6.2 The *ONME* as a MDD approach

With regard to **conceptual modeling patterns**, The *ONME* has adopted the well-known OMT strategy [Rumb98a] by dividing the conceptual modeling process into three complementary views: the object view, the dynamic view, and the functional model view (adding a fourth view to specify presentation patterns). When software engineers are specifying the system, what they are doing is capturing a formal specification of the system according to the *OASIS* formal specifications language [Past95]. This feature allows the introduction of a well-defined expressiveness in the specifications, which is often lacking in conventional methodologies.

The use of such a formal specification provides the context to validate the system in the *problem space*, obtaining a software product that is functionally equivalent to the specifications. This equivalence is achieved by creating a model compiler that implements all mappings specified between the conceptual patterns (*problem space* level) and their software representations (at the *solution space* level). Naturally, we have had to introduce relevant information to address specific features of *OASIS* in these diagrams

(Object Model, Dynamic Model, Functional Model, and Presentation Model). Nevertheless, this is always done preserving the external view that is compliant with the most extended modeling notation, which is the UML [Booc99].

Hence, the subset of UML used in the *ONME* is the one that is necessary to complete the information relevant for filling a class definition in *OASIS*. In this way, the arid formalism is hidden from the modeler when is describing the system by making it more comfortable to use a conventional notation. Another principal objective in the design of the *ONME* was to keep modelers from having to learn another graphical notation in order to model an information system. Having a formal basis allows us to provide a modeling environment where the set of needed diagrams is clearly established.

In the following, we briefly introduce the four conceptual model views that exist in the *ONME* approach.

6.3.1.1 Object model

The object model is a graphical model where system **classes** and **relationships** (association, aggregations, and inheritance) are defined. Additionally, **agent relationships** are specified to state the services that objects of a class are allowed to activate. These primitives capture the static point of view of the system. The corresponding the UML-based diagram is the Class Diagram, where the additional expressiveness is introduced by defining the corresponding stereotypes. Specifically, for each class, the Object Model captures:

- **Attribute:** to indicate whether the attribute is constant, variable or derived;
- **Services:** name of the services with their corresponding arguments, distinguishing the new and destroy class services. Also, a service definition can be included in the specification of more than one class, representing a synchronous communication mechanism between the object involved in their occurrence. They are called shared services and the participating services are linked with a double line.

- **Derivations:** derivation expressions for the derived attributes (those whose value is dependent on other attributes).
- **Constraints:** well-formed formulas representing conditions that objects of a class must satisfy.

The additional information associated with associations, aggregations, and inheritance is the following:

- For associated classes, to specify whether there is an aggregation or a composition (following the UML characterization) and whether the association is *static* or *dynamic*.
- For inheritance hierarchies, to specify whether a specialization is permanent or temporal. In the former case, the corresponding condition on constant attributes must characterize the specialization relationship; in the latter, a condition on variable attributes or the carrier service that activates the child role must be specified.

Finally, **integrity constrain** allows specifying conditions that must hold in any valid state of an object. They are specified within the class scope as well-formed formulas built on class attributes.

Figure 6.3 shows an example of an Object Model, a view of a library system with books, readers, and loans. Classes and their relationships are depicted using the *ONME* notation using the OlivaNova Modeler®.

For example, the discontinuous lines indicate *agent relationships*, in this model, there is one client class (Librarian) and the others are server classes. In the example, the objects of the librarian class can activate the services *new_reader*, *destroy_reader*, and *modify_reader* of the *reader* class.

The arrow between reader and *unrealible_reader* denotes that this class is a temporal specialization of *reader*. This occurs when the event *to_punish* is triggered. The solid lines between *reader* and *loan*, *loan* and *book*, *author* and *book*, and *book* and *supplier* represent an association between these classes. Finally, the double lines denote shared services. For instance, the service *to loan* is shared between the classes *reader* and *loan*.

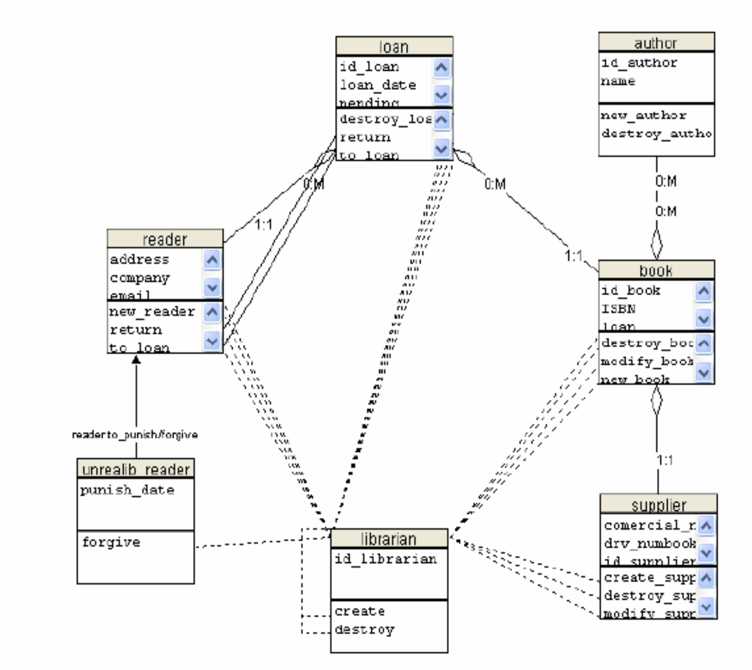


Figure 6.3 Class diagram for the library system

6.3.1.2 Dynamic model

The system class architecture has been specified using the Object Model. Additionally, basic features (such as which object life cycles can be considered valid and which inter-object communication can be established) have to be introduced in the system specification. To do this, the *ONME* provides a dynamic model. It uses two kinds of diagrams: State Transition Diagrams and Interaction Diagrams.

The **State Transition Diagram (STD)** is used to describe the correct behavior by establishing valid object life cycles for every class. By valid life, we mean appropriate sequence of service occurrences that characterizes the correct behavior of the objects that belong to a specific class. The corresponding UML based diagram is the State Diagram.

The syntax for transitions is the following:

*[list_of_agents| *] : [preconditions] service_name [WHEN control_condition]*

where services preconditions state what conditions must hold for activating and even and control conditions are conditions defined on object attributes to avoid the possible non-determinist for a given service activation.

Figure 6.4 shows a simple STD for the book class. Once a book is in the state labeled book(), if a *to_loan* service occurs and the precondition *available = true* is satisfied, the object will move to the *loaned* state. Here, if a *return* service occurs, the object will move to the *book*() state.

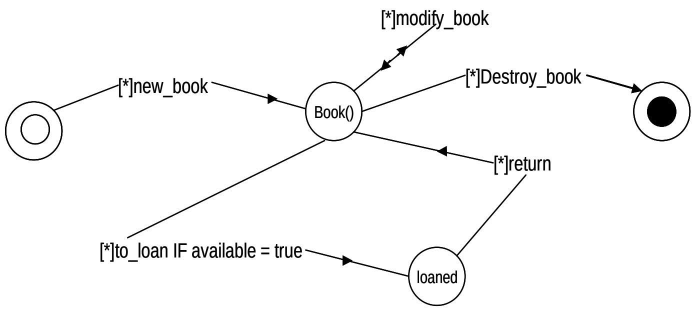


Figure 6.4 State transition diagram for the book class

Every service occurrence (i.e., *new_book*) is labeled by the agent *librarian*, which is allowed to activate it. In this example the "*" denotes that any valid agent class can activate the transition. As the only valid agents for the *new_book* service are objects of class *librarian*, both representations are equivalent.

In the example in Figure 6.5, once a *reader* is in the state labeled *WithouB* (Without books), if a *to_loan* service occurs, the object will move to the *WithB* (with books) state. Here, if a *return* service occurs, the selected transition will be the one that satisfies the corresponding control condition (n_books = 1 or n_books >1).

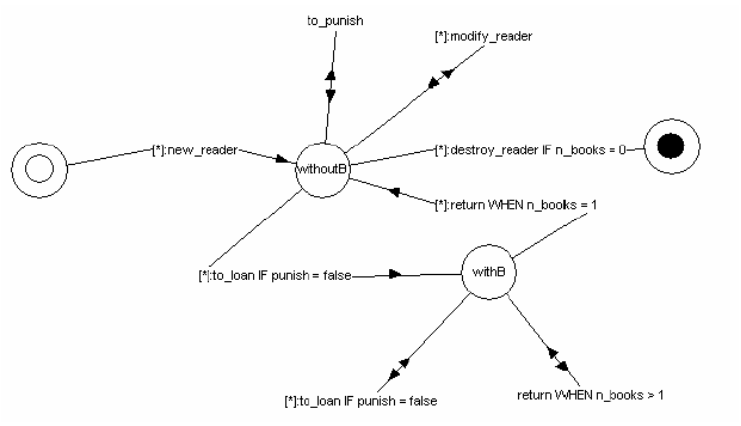

to_punish
[*]:modify_reader
[*]:new_reader
withoutB
[*]:destroy_reader IF n_books = 0
[*]:return WHEN n_books = 1
[*]:to_loan IF punish = false
withB
[*]:to_loan IF punish = false
return WHEN n_books > 1

Figure 6.5 State transition diagram for the reader class

The **Interaction Diagram (ID)** specifies the inter-object communication. We define two basic interactions: triggers, which are object services that are activated in an automated way when a condition is satisfied, and global interactions, which are transactions involving services of different objects. The corresponding UML base diagram is the collaboration Diagram where the context of the interaction is not shown.
Trigger specifications follow the syntax:

Destination:: (trigger_condition) agent:service

The first part of the formula is the destination (the object (s) to which the triggered service is addressed). The trigger destination can be the same object where the condition is satisfied (*self*), or a specific object (indicating the *oid*), or the entire class population if we are broadcasting the service (*class*). The last part of the formula is the triggered service with the corresponding agent.
Global interactions are graphically specified by connecting the services involved in the declared interaction. These services are represented as solid lines linking the objects (boxes) that provide them.

There is one STD for every class, but only one ID for the whole system, where all the inter-object interactions will be graphically specified.
For the library system, an interaction diagram with the following trigger can be defined:

Self:: (if (today() – loan.return_date) > 7) librarian:disable

It indicates that a trigger action (*librarian:disable*()) must be activated when the *return_date* is greater than 7 days.

6.3.1.3 Functional model

A correct functional specification is a shortcut of many of the most extended OO Methods. Sometimes, the model used breaks the homogeneity of the OO models, as happened with the initial versions of OMT, which proposed using the structured DFDs as a functional model. The use of DFD techniques in an object-modeling context has been criticized for being imprecise, mainly because it offers a perspective of the system (the functional perspective) that differs from the other models (the object perspective).
Other methods leave the free-specification of the system operations in the hands of the designer. The *ONME* functional model (FM) is quite different from these conventional approaches. In this model, the semantics associated to any change of an object state are captured as a consequence of a service occurrence. To do this, it is declaratively specified how every service changes the object state depending on the arguments of the service involved and object´s current state. A clear and simple strategy is given for dealing with the introduction of the necessary information. The relevant contribution of this functional model is the concept of *categorized attributes*.
Three types are defined: *push-pop*, *state_independent*, and *discrete-domain based*. Each type will define the pattern of information required to define its functionality.

- **Push-pop attributes** are those whose relevant events increase or decrease their value by a given quantity. Events that reset the attribute to a given value can also exist.

- **State-independent attributes** have a value that depends only on the latest action that has occurred. Once a relevant action is activated, the new attribute value of the object involved is independent of the previous one. In such a case, we consider that the attribute remains in a given state, having a certain value for the corresponding attributes.
- **Discrete-domain valued attributes** take their values from a limited domain. The different values of this domain model the valid situations that are possible for object of the class. Through the activation of carrier actions (which assign a domain value to the attribute), the object reaches a specific situation. The object abandons this situation when another event occurs (a “liberator” event).

Table 6.1 shows the Functional Model for the attribute *n_books* of the book class. This attribute is categorized as a *push-pop* because it’s a relevant service that increases or decreases the attribute value by a given quantity.

Table 6.1 Functional model example

Class:book	Attribute:n_book	Category: push-pop
increase	librarian:to_loan()	+1
decrease	librarian:return()	-1

In the example, *Librarian:to:to_loan()* is the increasing action and *librarian:return()* is the decreasing action. This categorization of attributes allows us to generate a complete *OASIS* specification in an automated way, where service functionality is completely captured.

6.3.1.4 Presentation model

The object society structure, behavior, and functionally are specified using the three conceptual models described above. The last step is to specify how users will interact with the system.
This is done by the Presentation Model [Moli03] through the definition of a set of Presentation Patterns. The Presentation Patterns capture the information required to characterize what appearance the

application will have, and how the user will interact with the application. These Presentation Patterns can be organized in three levels:

- **Level 1:** The *first level* contains the **Hierarchical Action Tree (HAT)** pattern, providing the access to the application.
- **Level 2:** The *second level* contains the *Interaction Units*. The user interface is the decomposed in the following scenarios to support user tasks:
- **Service Interaction Unit (SIU):** represents a dialogue where a user executes a service. As part of the dialogue, the user must provide the arguments of the service and the system must validate them. In addition, the user can perform an action to execute the service or to cancel it. The system will have to inform the users of the occurrence of errors.
- **Instance Interaction Unit (IIU):** the intention is to represent data (from one class instance) to the user. It is defined on a class and has the following properties: a visualization display (to show the information), actions (performed on the object) and navigation (reachability between instances).
- **Population Interaction Unit (PIU):** the intention is to present data (from a set of class instances) to the user. Filtering and ordering mechanisms can also be added to improve the object selection and consultation.
- **Master/Detail Interaction Unit (MDIU):** the intention is also to present information to the users. It is defined from IIUs and PIUs to show related information.
- **Level 3:** the third level is composed of patterns that add extra semantics to the interaction units.

The precise description of the pattern can be found in [Moli03]. Figure 6.6 gives an overall view of the levels and the corresponding interdependencies among patterns.

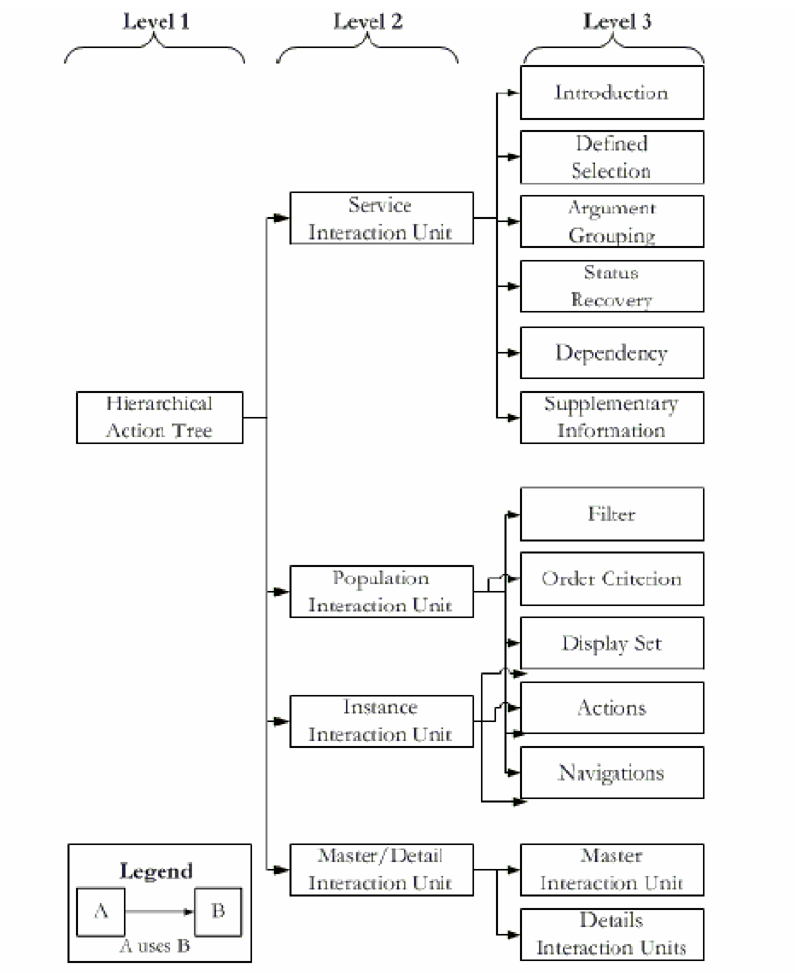


Figure 6.6 Pattern language for presentation model (source [Moli03])

6.4 The generation process of the conceptual model

This section describes our method for generating object-oriented conceptual schemas from an organizational model that is represented in the Tropos Framework. The entire methodology presented in this thesis consists of six processes that span the spectrum from Tropos business analysis to object-oriented conceptual modeling. We have made the application of the process systematic by proposing rules, patterns and guidelines for each one of the modeling stages.

Figure 6.7 shows an overview of our method. In this figure, a parallelogram represents an input of a process; the squares with rounded borders represent the processes of the proposed method, and the squares represent the deliverables of the defined processes.

The organizational diagrams specified in the Tropos framework (the actor diagram and the goal diagram) are the input of the first process: the ***goal-based requirements elicitation process***, which allows us to have a deep understanding of the organizational environment in order to identify the relevant tasks that should be automated to best satisfy the organizational goals. Goals are the key concept in this modeling stage where the current situation of the enterprise is represented using the social and intentional concepts of the Tropos framework. We argue that the representation of this situation is a fundamental stage when an existing enterprise is being modeled. One of the goals of this step is to determine the relevant actors in the organization. Later on, in this same step, the strategic organizational goals and their associated organizational plans are determined so that the internal behavior of the actors needed to satisfy their own goals and dependencies can be described. In this first process, which is explained in Chapter 3, design guidelines are used to represent quality attributes that the enterprise wants to improve by inserting a software system that automates some relevant organizational processes.

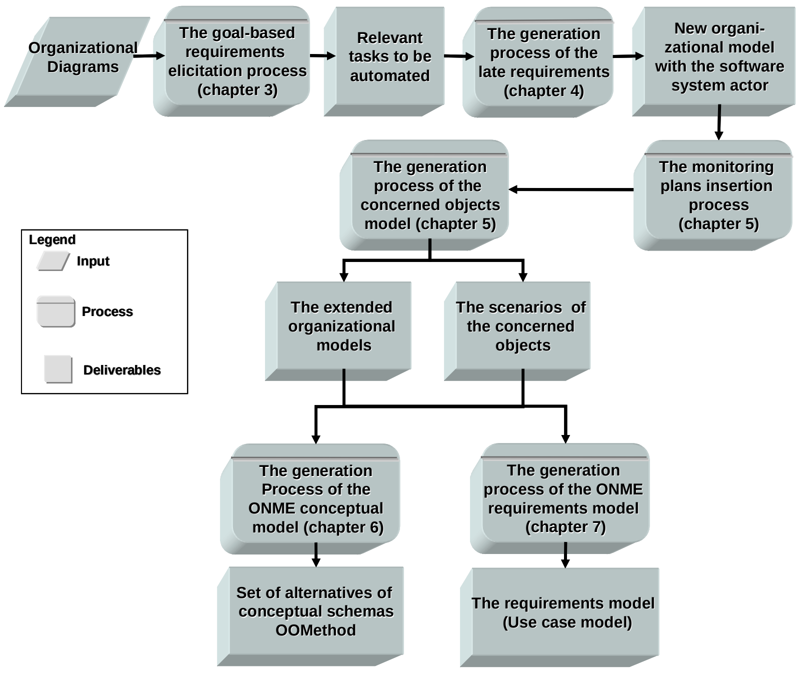

Organizational Diagrams
The goal-based requirements elicitation process (chapter 3)
Relevant tasks to be automated
The generation process of the late requirements (chapter 4)
New organi-zational model with the software system actor
The monitoring plans insertion process (chapter 5)
The generation process of the concerned objects model (chapter 5)
Legend
Input
Process
Deliverables
The extended organizational models
The scenarios of the concerned objects
The generation Process of the ONME conceptual model (chapter 6)
The generation process of the ONME requirements model (chapter 7)
Set of alternatives of conceptual schemas OOMethod
The requirements model (Use case model)

Figure 6.7 Overview of the method to link business and system specifications

In the second process (*the late requirements generation)*, which was explained in Chapter 4, we carried out the delegation of the relevant plans to be automated towards a new organizational actor that represents the software system. It is important to point out that this process of creation of a new organizational actor that represents the system-to-be is one of the main contributions of this work. In current research works that focus on using the organizational context to generate software requirements specifications [Alen03] [Jian06] [Sant02], the requirements model has the organizational model as source, which is directly extended to represent domain concepts.

We consider that there are certain aspects that need to be also taken into account in order to make the transformation between organizational model and software specifications possible:

- First: Not all the organizational plans are candidates to be automated; therefore, if there is no mechanism to isolate the relevant information, all the information about plans to be manually executed and plans to be automated are mixed in the same model.
- Second: One of the main issues of the Tropos framework is the that when large enterprises are modeled, the models are overcharged containing a large number of modeling elements in the same diagram. This can be even worse if the diagram also includes the domain information needed to determine the plans to be automated.
- Third: By inserting the software system actor (SSA) in the organizational model, it is possible to illustrate the actors that will interact with the system-to-be, and it is also possible to determine the nature of the user interactions.
- Fourth: By inserting the SSA in the organizational model, it is possible to focus the analyst's efforts on the definition of the plans that the system must perform based on the plans defined within the system actor.

In our approach, the relevant goals and plans are delegated to the SSA to indicate that this actor is the new one that is responsible for fulfilling these goals and plans. One of the advantages of this approach is that it is possible to take design decisions based on current organizational goals and plans instead of starting the design process from scratch. The goal and plan delegation is guided by a pattern language that represents all the possibilities that exist for reducing the abstraction level of a *"pure"* organizational model to be closer to a software requirements specification. As a result of this process, a new organizational model that includes the SSA is created, which represents the expected functionality of the system-to-be.

In the third and fourth processes (*the monitoring plans insertion process* and *the generation process of the concerned objects model*) the organizational model is extended to generate a model that shows the relevant plans to be automated. These processes, which were explained in Chapter 5, enable the analyst to identify the scope of the system-to-be.

The resultant models obtained in these processes are the *extended organizational models* and the *concerned object scenarios*. These elements are used in the fifth process, *the generation process of the ONME conceptual model*, which generates a space of alternative object-oriented conceptual schemas that fit the business functionalities. These elements are also used in the sixth process: the *generation process of the ONME requirements model* (presented in Chapter 7), which uses the extended organizational model with the SSA to generate a specification of the requirements for the system-to-be.

It is important to point out that in this thesis both approaches (the generation of conceptual schemas from organizational models and the generation of requirements models from organizational models) have been proposed as alternative solutions for the problem of associating business and software system descriptions.

The process for carrying out the *ONME* conceptual model generation is presented below in detail.

Overview of the *ONME* conceptual model generation

In this section, an overview of the proposed method to generate the object-oriented conceptual model is presented (Figure 6.8).

One of the main contributions of our work is the possibility to generate a space of alternative conceptual models that reflect a specific view of the structure of the system-to-be. The generation of the space of alternatives consists in a progressive method that is based on the analysis of relevant organizational goals and non-functional requirements (represented as *softgoals* in Tropos).

The idea of generating conceptual schemas according to a specific criterion is not a new idea; it was introduced in the 70´s to create normalized database schemas. In [Codd79] [Chen76] [Saka80] [Shar02] [Stee96] several criteria were defined to guide the definition of database schemas that fit a specific criterion.

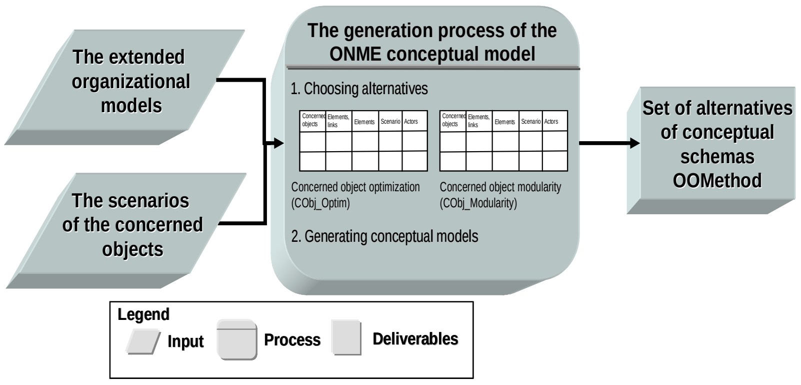

The extended organizational models
The scenarios of the concerned objects
The generation process of the ONME conceptual model
1. Choosing alternatives
Concerned objects
Elements, links
Elements
Scenario
Actors
Concerned object optimization (CObj_Optim)
Concerned object modularity (CObj_Modularity)
2. Generating conceptual models
Set of alternatives of conceptual schemas OOMethod
Legend
Input
Process
Deliverables

Figure 6.8 The schema of the generation process *of the ONME* conceptual model

However, in object-oriented conceptual modeling, where the conceptual model represents the database and the system structure, there are only a few research attempts to create normalized schemas to fit a specific criteria.

The inputs of this process of generating the space of alternatives are the extended organizational models and the table of scenarios of the concerned objects. The extended organizational models are those that consider monitoring plans and those where the concerned objects have been identified.

As stated above, this process is guided by a set of algorithms and rules, which has been grouped in two sub-processes:

- Generation of a space of alternatives
- Generation of conceptual models

In the following sections, these two processes to generate the conceptual model for the system-to-be are presented in detail.

6.4.1 Generation of a space of alternatives

The first step to generate a conceptual schema for the system-to-be is the generation of the space of alternatives. To do this, the lifetime of each concerned object (identified in the generation process of the concerned objects illustrated in Chapter 5) must be analyzed. To do

this, we analyze the different scenarios in which a concerned object has been manipulated throughout its lifetime.
By scenario, we mean some point in the organizational process where a concerned object is queried or modified.
A concerned object can generate several scenarios in order to represent the behavior of the relevant objects in the business from a temporal perspective.
The scenarios of the concerned objects are represented using a table of scenarios (Table 6.2). The concerned objects are stored in this table with all their characteristics. Each row of the table represents a scenario where a concerned object is used.

Table 6.2 Scenario of concerned objects (CObjs)

Concerned object name	Elements and asso-ciated links	Associated elements	Used attributes	Related actors	Label of the concerned object

For instance, Figure 6.9 illustrates the lifetime of the *Customer* concerned object of the *Car Rental* case study. The lifetime is composed of states in which the concerned object is used throughout the lifetime of the software system-to-be.

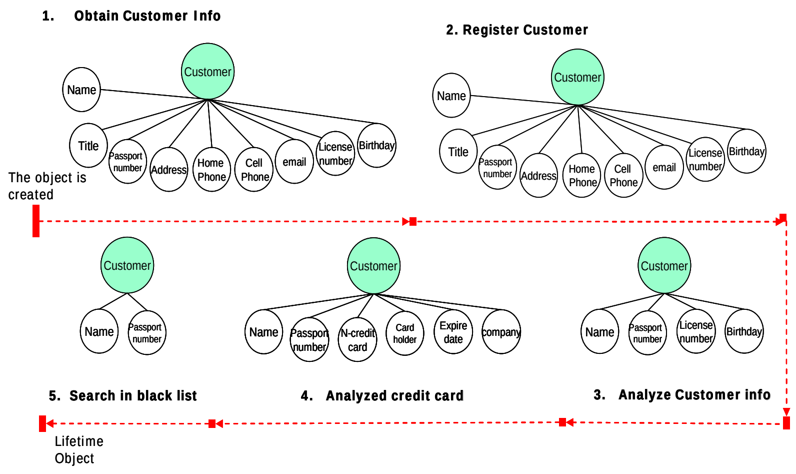


Figure 6.9 Lifetime of the *Customer* concerned object

First, the *Customer* concerned object is created when the plan *Obtain Customer info* is executed. The next stage of the concerned object occurs when the plan *Register-Customer* is executed.

Then, the plans *analyze Customer, analyze credit card, and search in black list* also modify the *Customer* concerned object. It is important to point out that a different set of attributes of the concerned object is used in the different stages where the object is manipulated.

Later on, the generated concerned objects must be organized depending on each alternative solution.

The analysis of the different stages of the concerned object enables the analyst to obtain information about semantic associations among attributes and also to obtain information about its usability:

- Identify the attributes that always appear in each state of the concerned object. For instance, in the *Customer* concerned object, the attributes: Name and Passport number appear in all the object states (Figure 6.9). These kinds of attributes are candidates to be considered as identifiers of the candidates classes.
- Identify those attributes that are rarely modified in the organizational process. For instance, the attributes of the plan

Register-Customer: *Title, Address, Home-Phone, Email* and *Birthday*. These attributes are only used in the creation of the concerned object; therefore, they were not used in the following stages of the concerned object. These kinds of attributes are candidates to be considered as attributes of classes in a conceptual schema that promotes optimization criteria.

- Discover the semantic relationship among attributes, this gives the analyst the possibility of encapsulating closely related attributes in new classes based on cohesion criteria. For instance, Figure 6.10 shows the attributes *engine*, *number of cylinders* and *number of displacements*, which can be grouped into a new concerned object *engine*. Therefore, a new class with the associated attributes can be created to isolate these elements.

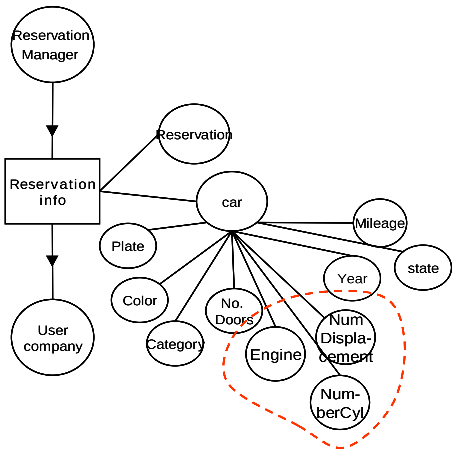


Figure 6.10 Attributes in two different concerned objects

The selection of a specific criterion is the basis for determining the appropriate translation schema of attributes of this kind.

We have selected two different solutions, which are represented as conceptual schemas, for the running example. The solutions are based on the following criteria: optimization and modularity. As stated above, the definition of conceptual schemas according a specific criterion has been analyzed in depth in several research works [Dull03] [Shar02] [Stee96] [Ambl03] [Bock97] [Desa07]. In these works, there are standard solutions for creating a conceptual model that appropriately fits the selected criteria.

In the following subsections, each alternative solution for the running example is detailed. The generation of solutions according to a specific enterprise criterion is guided by algorithms that permit to represent the information of each concerned object to be represented in order to satisfy the organizational goal.

It is important to point out that that the generation of different conceptual schemas according to quality criteria affect not only the definition of the database design, but also the generated software system. However, the process to create new conceptual schemas is based on the static part of the class diagram (class attributes and relationships among classes), and it does not consider the dynamic part of the schema (class methods). This is the reason why the generation of alternative schemas is a data-based optimization process.

6.4.1.1 Optimization strategy for conceptual model generation

A definition of optimization in computer science is the following: *"improving a system to reduce runtime, bandwidth, memory requirements, or other property of a system; in particular"* [WOpt07]. In database theory, optimization is analyzed in the optimization of access paths and the storage of data in the file system level [Shar02]. In conceptual modeling, optimization is regarded as reducing the semantic distance among closely related attributes.

The strategy to create a global model that promotes optimization is to create global classes in the conceptual model. This is to reduce the response time when a class is queried or stored in a database of the system. Therefore, the strategy to create a conceptual schema for optimization consists of concentrating all the attributes as close as possible in order to reduce the access to different databases when a query is executed.

The algorithm for reconciling concerned objects according to the strategy of optimization is illustrated below. The algorithm is guided by the function ***Reconciling_optimization()***, whose aim is to create a complete concerned object. By complete concerned object, we mean the concerned object that concentrates (as close as possible) all the attributes that belong to the object throughout its lifetime.

The strategy of this process consists of creating a new table of scenarios that contains the complete concerned objects. This table will contain the attributes of the concerned objects grouped according to their semantic distance in order to create global concerned objects.

Table 6.3 depicts the structure of the complete concerned objects table. The first column (***Concerned object)*** contains the name of the concerned object that is identified; the second column (***Used attributes)*** contains the attributes of the concerned object that is identified according to the solution criteria (Optimization or Modularity); finally, the third column (***Label of the Concerned object)*** contains the current situation of the concerned object (for instance, if a concerned object was divided between other concerned objects).

Table 6.3 Complete concerned objects Table (CObj_Optim)

Concerned object	Used attributes	Label of the concerned object

In this solution criterion, the concerned objects being analyzed must be placed in a new table called ***CObj_Optim,*** which will contain the list of complete concerned objects grouped by the Optimization criteria.

Next, the selected concerned objects must be compared with the rest of the concerned objects in all scenarios tables (***CObjs*** table). When other scenario of the same concerned object is found, the attributes of both concerned objects must be compared and unified by including the attributes in the ***CObj_Optim*** table. The idea is to create a table that mixes all the attributes for each concerned object. The function ***Insert Mix-attributes()*** performs the match among attributes, and it also executes the creation of the new table ***CObj_Optim***. Once a concerned object has been analyzed in the scenario table, it must be labeled in order to indicate that it has already been analyzed.

The ***Insert COAppropriate ()*** function is used to insert a concerned object in the *CObj_Optim* table with all the attributes that were found in the lifetime of the concerned object.

For example, Figure 6.9 shows the lifetime of the *Customer* concerned object. Once the algorithm for reconciling concerned objects by optimization criterion is applied, the *Customer* concerned object shown in Figure 6.11 is obtained. This complete concerned object contains the following attributes: *Name, title, passport number, address, home-phone, cell phone, email, license number, birthday, N-credit card, card holder, expire date* and *company.* Thus, according to the optimization criterion, we obtain a concerned object makes up of all its attributes that were manipulated in its lifetime.

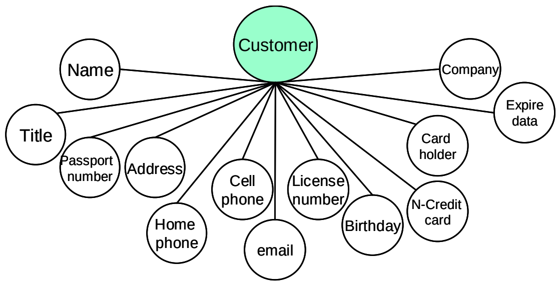


Figure 6.11 An example of applying an algorithm to reconcile concerned object by the optimization criterion

The algorithms that perform the mapping among attributes and the mixture of attributes are presented below. It is important to point out that only the main algorithm is presented in detail. The secondary functions are only described in the text. Table 6.4 shows the complete concerned object table for the object *Customer.*

Table 6.4 The complete concerned objects table for the object *Customer*

Concerned object name	Used attributes	Label of the Concerned object
Customer	Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number,birthday, N-Credit card, Card holder, expire data, Company	

Algorithm for reconciling concerned objects by its optimization

```
Reconciling_Optimization()
Begin
     i=1, k=1, N; //Num de scenarios of concerned objects
     While (i<= N) do
         If (Table CObjs [6][i] != “Analyzed”)
              Insert COAppropriate (Table CObjs, Table COb_Optim,
                                    i, k, Label)
          Table CObj_Optim[1][k] = Table CObj[1][i]; // insert the first
      // column in the table of scenarios “Concerned object name”
           Table CObj_Optim[2][k] = Table CObjs[4][i]; // insert the second
      // column in the table of scenarios “Used attributes”
      Table CObj_Optim[3][k]= Label ;
       J= i+1;
              While (j < N+1)
                    If (CObj1[1][i] == CObj2[1][j]) then
                          Insert Mix-attributes (Table CObjs,
                                  Table  CObj_Optim, i,j, k, Label)
                    End if
                    Else j=j+1;
              End while
        End if
        Else i=i+1;
End Reconciling_Optimization
```

Insert Mix-attributes Function

(Table CObjs, Table CObj_Optim, i,j, k, Label)

The main objective of this function is to compare the attributes stored in the different scenarios for each identified concerned object (*CObjs* table) and to join all the attributes of a concerned object in the *CObj_Optim* table. This table must contain all the complete concerned objects in accordance with optimization criteria.

This function has the following elements as function parameters: the CObjs table, which is analyzed in the following positions: column 4, row *i*, and column 2, row *j*. The positions are analyzed in order to determine if the attributes of both concerned objects are the same or if some differences among attributes are detected. If there is an attribute with a semantic difference, then it must be inserted in the *CObj_Optim* Table in the position column 2, row *k*. Finally, the "***Analyzed***" label must be inserted in column 6, row *j* of the *CObjs* Table to indicate that this element has already been analyzed.

When a set of concerned objects has been grouped into a concerned object, then, the "***Grouped***" label must be included in the attributes of the objects of the CObj_Optim Table along with the name of the concerned object that gives rise to the new concerned object.

6.4.1.2 Modularity strategy for conceptual model generation

Modularity is the property of computer programs that measures the extent to which they have been composed of separate parts called modules [WMod07]. We consider *encapsulation* as the most important quality of modularity.

The strategy to create a conceptual model that promotes modularity consists of analyzing the use of the attributes of the concerned objects throughout their lifetime in order to detect the information that is used most frequently and the information that is very rarely used. Based on this information, the data object can be encapsulated according to this usability criterion. The idea of this process is to implement the separation of concerns.

As stated above, a concern expresses a specific interest in some topic pertaining to a particular system of interest (or other subject matter) [Hill99]. In this sense, one of the most rapidly emerging technologies in software engineering is the separation of concerns, which is an established software engineering theory based on the notion that it is beneficial to break down a large problem into a series of individual problems or concerns. This allows the logic required to solve the problem to be decomposed into a collection of smaller, related pieces. Each piece addresses a specific concern where usability is the selection feature.

The procedure to perform the optimization by modularity can be summarized in the following steps:

Step 1. Use the *CObject_Optim* table (which was explained in the previous section) to carry out the analysis of concerned object attributes.

Step 2. Compare the scenarios of each concerned object with the *CObj_Optim* table. This is done to determine the use of the attributes throughout the lifetime of each concerned object.

Step 3. Analyze the concerned object attributes in order to determine their usability. In this step we need to analyze the object attributes to find those that are rarely used throughout the lifetime of the concerned objects.

Step 4. Based on this criterion, the attributes that share usability characteristics are used to create new classes in the object model. The information about the concerned objects must be stored in a new table called *CObj_Modularity*.

Step 5. Label the new concerned objects to indicate that a concerned object was divided according to the usability criterion to create two separate classes.

An example of this strategy is shown in Figure 6.12. The concerned object *Car* has some attributes that are rarely used in the scenarios of this concerned object. The attribute *engine* is only modified when the *Car* is registered. Therefore, this concerned object must be divided in two different concerned objects.

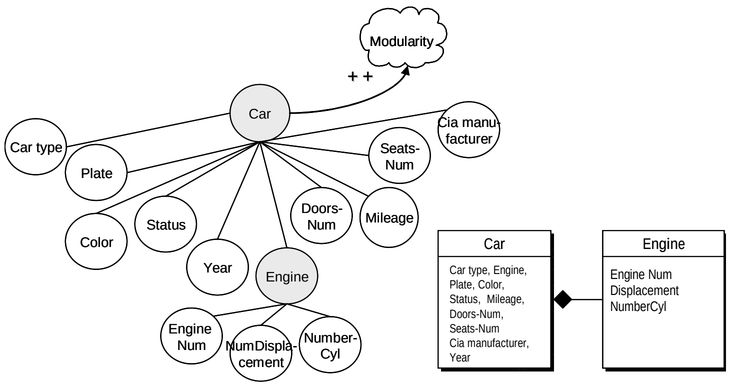


Figure 6.12 Concerned object divided into another concerned object

The algorithm for reconciling concerned objects by the strategy of modularity is presented below. The algorithm is implemented by the function ***Reconciling_Modularity(),*** which encapsulates concerned objects taking into account this usability throughout their lifetime. To do this, the ***CObj_Optim*** table is used. First, the number of concerned objects is obtained; then each attribute must be individually analyzed and compared with the table of scenarios of each concerned object (*Cobjs* table). This is done to obtain a temporal list of each attribute and the scenarios where it is used. This process is done with the function called ***Clasify_attributes()***.
The ***Reconciling_attributes()*** function determines the attributes that are rarely used and those attributes that are frequently used together. Therefore, a new concerned object will be created based on this criterion by dividing an entire concerned object into small fragments of information that are associated to a specific feature. This is done to isolate (in different modules) the information that is used frequently from the information that is rarely used.
A table (similar to the *CObj_Optim* Table) must be created to store the new concerned objects, or those that have been modified (the elements modified are those concerned objects that have been divided into different objects according to modularity criteria, therefore reducing the number of attributes). The table is called: ***CObj_Modularity***.

The algorithms that perform the mapping among attributes and the separation of attributes are presented below. It is important to point out that only the main algorithm is presented in detail. To facilitate the reading of the method, the secondary functions are only described in the text.

Algorithm for reconciling concerned objects by its MODULARITY

```
Reconciling_Modularity ()
Begin
     i=1, k=1
     N; //Num de concerned objects
     While (i<= N) do
         NAttr =  Obtain_Num_Attributes(i)
         While (j <= NAttr)
              Attribute= Object_Optim[4][i].J
             Clasify_attributes(Attribute)
           End while j
             ReconcilingAttributes()
    End while i
End Reconciling_Modularity
```

Clasify_attributes(Attribute)

The main objective of this function is to compare an attribute with each scenario where it appears. The result of this function is a temporal table with all attributes information. The attribute to be analyzed is the parameter of this function.

ReconcilingAttributes()

This function uses the temporal table created in the classify_attributes function. Therefore, these attributes must be analyzed to determine if an attribute is frequently or rarely used and which attributes generally appear together with the analyzed attribute. These attributes must be joined to create a new concerned object. The information is placed in a table similar to *CObj_Optim*, but in this case, it will be called: ***CObj_Modularity*** table. The attributes that have generated a new concerned object must be labeled with the word "***Divided***", along with the name of the concerned objects that were affected in the attribute separation process.

6.4.2 Rules for generating the conceptual model

The generation of the *ONME* object-oriented conceptual schema is a process that is based on the space of alternatives determined in previous steps. The idea is to define a specific conceptual model according to the feature selected by the stakeholders. Therefore, in this process, the tables generated from the conceptual model (*CObj_Optim* table, *CObj_Modularity* table) are the basis for the generation of the conceptual schema.

It is important to point out that the transformational rules for generating the conceptual schemas are independent from the alternative selected for optimizing the concerned object schema.

Before defining the rules to generate the conceptual model, an overview of the specific sources for each fragment of the *ONME* conceptual schema is presented below.

The *ONME* conceptual model is composed of an object model, a dynamic model, a functional model, and a presentation model. We focus on the object model that represents the data and the static structure of the system-to-be. As state above, the extended organizational models and the tables of the scenarios of the concerned objects[1] are the bases for the generation of the conceptual schema.

The process performs an analysis of the extended organizational models and tables of the scenarios of the concerned objects looking for the constitutive UML-based elements of an object model: classes, services, attributes, integrity constraints, associations, aggregations, inheritances.

- **Classes** are generated from the concerned objects specified in the table of scenarios of the concerned objects (**CObj_Optim table or CObj_Modularity table**).

1 These tables will be the different alternatives for a solution, where the concerned objects can be represented.

- **Services** are generated from the elements associated to the concerned object in the table of scenarios (**CObjs** table).
- **Attributes** can be obtained from the **Used attributes** by the analyzed concerned object in the **CObj_Optim table or CObj_Modularity table**.
- **Associations and aggregations** can be obtained by analyzing the plans, resources or goals in the organizational model. The aggregation can also be obtained looking for concerned objects with the label "**divided**" in the table of scenarios.
- **Inheritance relationship** can be obtained from extended organizational models when a relationship of **plays** (which represents the roles of an actor) between actors is defined (only if they are considered as concerned objects in the extended organizational model.)

6.4.2.1 Rule for identifying a class

Rule 1: For every concerned object identified in each space of alternatives (*CObj_Optim* table, *CObj_Modularity* table), a class will be generated in the class diagram.

Applying rule 1 to the partial view of the table of scenarios of the concerned objects (Figure 6.13), the classes that can be obtained are: *Customer*, *Extra service* and *Reservation*.

a) Table of scenarios of the concerned objects

Concerned Object	Element and link associated	Associated Element	Used attributes	Related Actors
Customer	Resource Dependency (Customer info)		Customer name, Title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, Birthday	CRS-User Company
Extra service	Resource Dependency (Data extras info)		Extra services type, Pickup date, Return date	CRS-User Company
Reservation	Sub plan AND decomposition (Obtain reservation data)	General Plan (*Search car availability*)	Car type, Pickup Date, Return Date, Pickup Zone	CRS

b) Classes generated

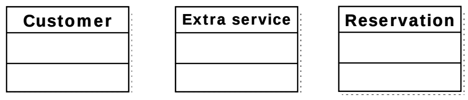


Figure 6.13 Example of classes generated for *Car Rental* case study

6.4.2.2 Rules for identifying attributes

Rule 2: All the attributes of a concerned object will be considered as attributes of the created class in the class diagram.

Rule 3: The type of the attributes will be identified by analyzing the scenarios of the concerned objects.

Rule 3.1: If the initial value of a concerned object’s attribute remains unalterable in all the scenarios where the attribute appears, then the type of this attribute must be *CONSTANT,* and it must be generated in the class diagram.

Rule 3.2: If the value of a concerned object’s attribute is manipulated in some pre- or post- condition of a plan or resource in extended organizational models, then the type of this attribute must be ***DERIVED,*** and it must be generated in the class diagram.

Rule 3.3: If the type of an attribute is not CONSTANT or DERIVED, then the type of the attribute will be ***VARIABLE***, and it must be generated in the class diagram.

An example of the generation of the attributes for a class is shown in Figure 6.14, which is the result of applying rules 2, 3 and 3.1 to the table of scenarios of the resource dependency *Info Customer* to obtain the class *Customer*. The attributes of the *Customer* class are *Constant* since they remain unalterable in the other states where they appear.

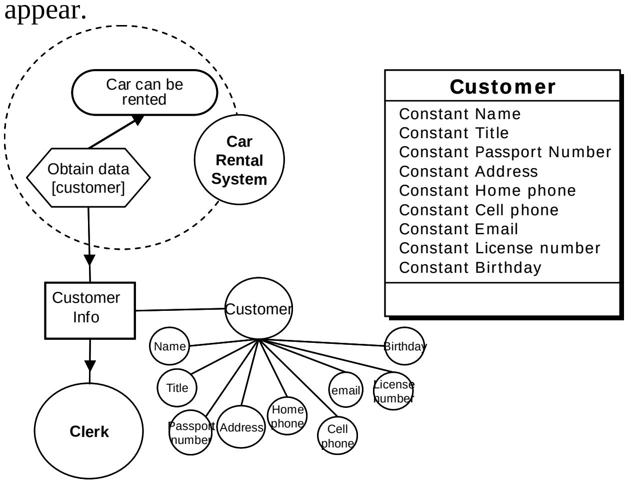


Figure 6.14 Example of attributes generated for the class *Customer*

6.4.2.3 Rules for identifying Events and Transactions

Rule 4: For each final plan or plan dependency in the extended model, one or more events will be generated in the class diagram.

Rule 4.1: If the analyzed plan handles only one concerned object, then an event in the class diagram will be created. The event will be placed in the class that is generated from the concerned object.

Rule 4.2: If the analyzed plan handles a concerned object that has the *divided*[1] label, then an event in each class of the concerned object that is generated must be created; the type of the created events will be *Shared*.

[1] The label "*divide*" is placed on those concerned objects that have been divided into other objects in the reconciling phase.

Rule 4.3: If the analyzed plan handles two or more concerned objects (after the algorithm for reconciling concerned has been applied), then it will be translated into an event of type *Shared* in the classes that are generated by the concerned objects of the analyzed plan.

In the event generation process, the *New* and *Destroy* events need to be elicited by the requirements engineers. To do this, the scenarios of the concerned objects must be taken into account to determine the state of the object when it is created and the state of the objects when the lifetime of the object is completed. It is important to point out that both *New* and *Destroy* events imply user interaction with the software system. This is the reason why these events must be generated from plans that are associated to dependency relationships. In Figure 6.15, we present an example of the final plan *obtain data*, which generated the *Customer* concerned object. The event *obtain_data* is created in the class *Customer* by applying the rule 4 and 4.1.

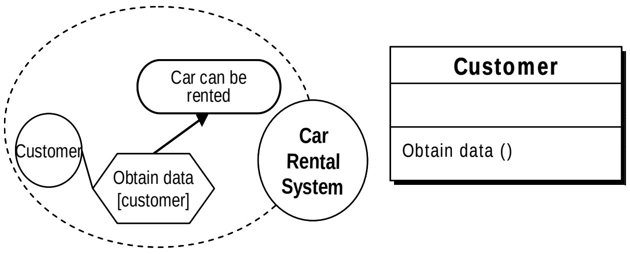


Figure 6.15 Example of an event generated for the class *Customer*

Rule 5: For every plan with an *AND* decomposition link, a transaction must be created. As stated above, a *composite* plan is decomposed into a set of low-level plans to make it operational. Thus, the decomposition implies a strong dependency among the root and the leaf nodes. If the leaf nodes are performed, then the root is also performed. The decomposition is translated into a transaction in the class that is created from the concerned object associated to the root node.

Rule 6: For every plan or goal with an *OR* decomposition link, a set of alternative methods must be created in the classes generated from the associated concerned object. As stated above, a *composite* plan can be decomposed into a set of alternatives plans, where the satisfaction of any one of the leaf nodes (that represent the alternatives) fully satisfies the parent node. Therefore, this decomposition is translated into events in the class that is created from the concerned object associated to the root node.

6.4.2.4 Rules for identifying the association relationships

Rule 7: For every plan, resource or goal that handles two o more concerned objects, an association relationship between the generated classes of these concerned objects must be created. The reasoning behind this rule is that, in a plan that creates two or more concerned objects, these objects are usually closely related. However, no indications are given in the model to determine if the relationship can be modeled by an association or an aggregation. For this reason, we have selected the less restricted relationship (association) as the default option. However, the analyst can decide to indicate an aggregation in place of an association based on the strength of the relationship among the concerned objects, the visibility attributes, the existence dependency, reflexivity, symmetry and the delete propagation schema [Albe03]. An example of the generation of association relationships from concerned objects is shown in Figure 6.16.

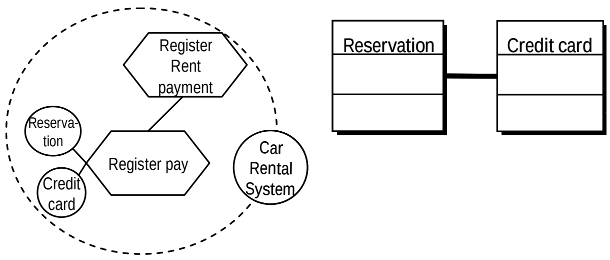


Figure 6.16 Example of an associated relationship

6.4.2.5 Rules for identifying the aggregation relationships

Rule 8: For every concerned object with the *divided* label, an aggregation relationship between the generated classes of these concerned objects will be created. It is important to point out that the concerned objects that are labeled as *divided* are the result of dividing a single class into two highly cohesive classes based on usability criteria. Therefore, a strong relationship exists among the divided classes. An example of an aggregation relationship of a divided concerned object is shown in Figure 6.12.

6.4.2.6 Rules for identifying the cardinality

Rule 9: To identify the cardinality of the associations and aggregation relationships, the extended organizational models need to be analyzed. The concerned objects that are the source of the classes need to be analyzed to determine the number of occurrences in which two specific concerned objects appear together in an association or aggregation relationship. However, it is important to point out that this analysis only gives preliminary results to obtain the cardinality in the relationships among classes in the class diagram.

6.4.2.7 Rules for identifying the inheritance relationships

Rule 10: The *play* relationship of the Tropos framework is used to create inheritance relationships. The *play* relationship indicates the existence of generic actors that play different roles in the organizational process. To define the inheritance relationships, it is necessary to determine if the generic actor and the corresponding roles have generated concerned objects, if so, an inheritance relationship needs to be defined between the classes that were originated from the actor and its corresponding role. Figure 6.17 shows an example of the generation of inheritance relationships from role relationships.

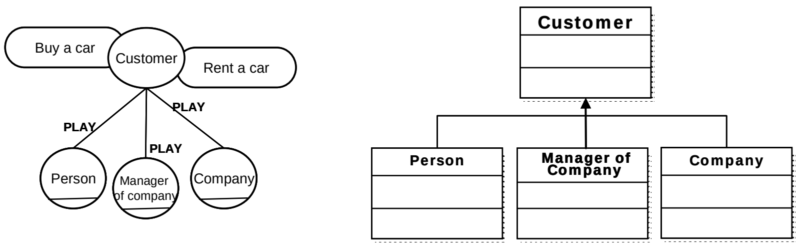
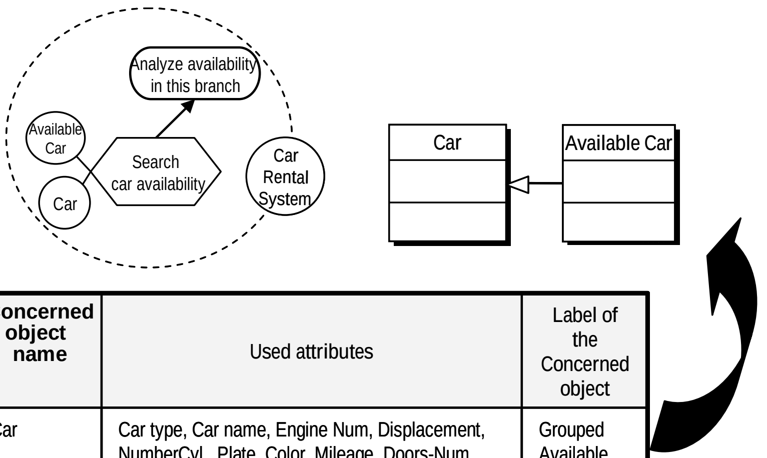


Figure 6.17 Example of an inheritance relationship

Rule 11: In order to define inheritance relationships, we need to look for concerned objects that are labeled with the "*grouped*" label. The inheritance relationship will be created between the generated class and the class that was the original source of the generated class. Figure 6.18 shows an example of the generation of an inheritance relationship from two concerned objects, where the optimization criterion was grouped in only one concerned object.



Concerned object name	Used attributes	Label of the Concerned object
Car	Car type, Car name, Engine Num, Displacement, NumberCyl , Plate, Color, Mileage, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day, Pickup Date, Return Date, Pickup Zone, Branch	Grouped Available Car

Table of appropriate concerned objects by optimization criteria

Figure 6.18 Example of an inheritance relationship

6.4.2.8 Rules for identifying triggers and cardinality restrictions

Rule 12: The pre- and pos-condition of the monitoring plans can be used to identify triggers. This is because these kinds of plans are used to observe the execution of the organizational tasks. This is done to define the current behavior model and correct the undesirable behaviors. When an undesirable event occurs, then a set of events needs to be activated to allow the system to recover from the failure. This kind of semantics can be appropriately modeled using triggers in the system-to-be. The pre- and post-conditions of the monitoring plans need to be analyzed to find the conditions that start the trigger.

An example of the conceptual model of the *Car Rental* case study

The extended goal model shown in Figure 6.19 is used to generate the *ONME* conceptual models. This model is a partial view of the *Car Rental* case study. The numeration of each element in the model is used to facilitate the example. The main goal of the model is *rental car management,* the subgoals derived from the main goal are: *Provide information about the car, Make reservations, Handover car, Return the car, Manage other branches, Cancel reservations, Query reservations and Modify reservations*. The first two goals are described in more detail. The concerned objects identified in each element are also depicted.

Table 4.4 shows the scenario table for all the concerned objects identified from the model in Figure 6.19. In this table, the lifetime of each concerned object is represented for analysis purposes. The scenario tables are the input for the process of generation of alternative conceptual models based on optimization or modularization criteria.

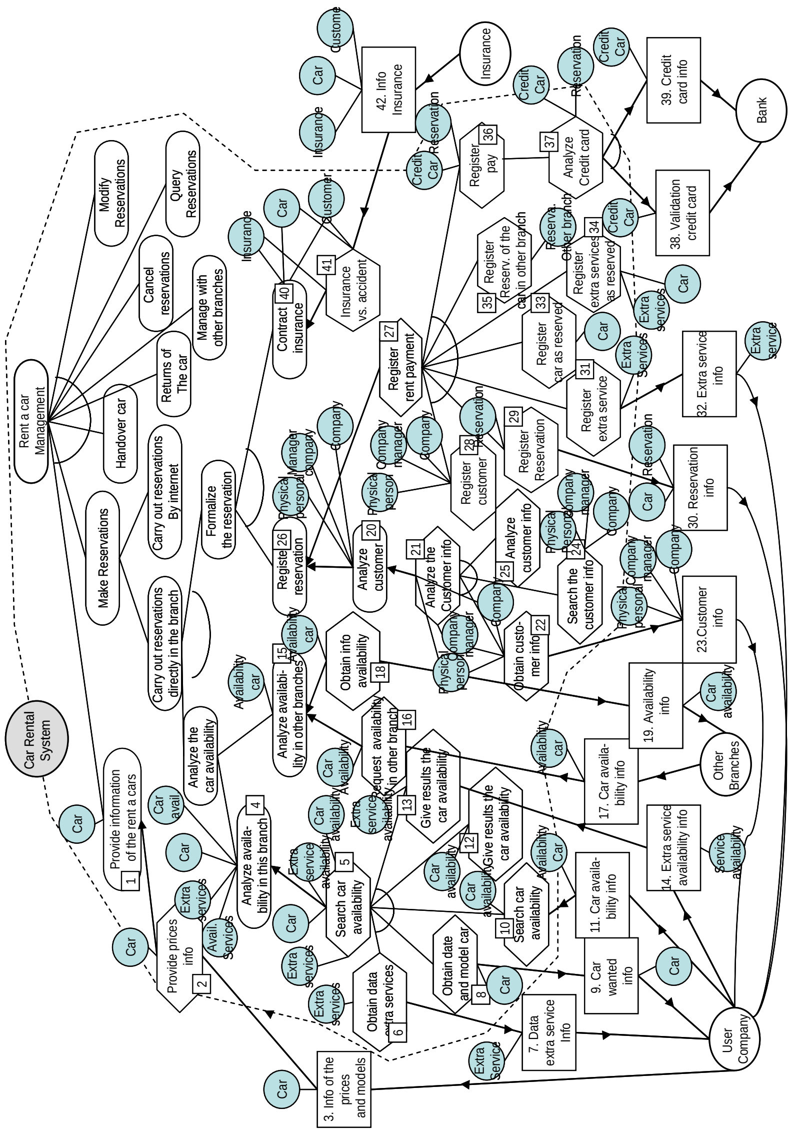

Car Rental System
Rent a car Management
Make Reservations
Modify Reservations
Query Reservations
Cancel reservations
Manage with other branches
Returns of The car
Handover car
Carry out reservations By internet
Carry out reservations directly in the branch
Formalize the reservation
Analyze the car availability
1 Provide information of the rent a cars
2 Provide prices info
3. Info of the prices and models
4 Analyze availa-bility in this branch
5 Search car availability
6 Obtain data extra services
7. Data extra service Info
8 Obtain date and model car
9. Car wanted info
10 Search car availability
11. Car availa-bility info
12 Give results the car availability
13 Give results the car availability
14. Extra service availability info
15 Analyze availabi-lity in other branches
16 Request availability in other branch
17. Car availa-bility info
18 Obtain info availability
19. Availability info
20 Analyze customer
21 Analyze the Customer info
22 Obtain custo-mer info
23.Customer info
24 Search the customer info
25 Analyze customer info
26 Register reservation
27 Register rent payment
28 Register customer
29 Register Reservation
30. Reservation info
31 Register extra service
32. Extra service info
33 Register car as reserved
34 Register extra services as reserved
35 Register Reserv. of the car in other branch
36 Register pay
37 Analyze Credit card
38. Validation credit card
39. Credit card info
40 Contract insurance
41 Insurance vs. accident
42. Info Insurance
Insurance
Bank
Other Branches
User Company
Car
Customer
Extra services
Availability car
Company
Company manager
Physical person
Reservation
Credit Car
Reserva. Other branch
Service availability

Figure 6.19 Partial view of the concerned object model of the *Car Rental* case study

Table 6. 5 Table of scenarios of Figure 6.19

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
1. Car	Goal (Provide information of the rent a cars)		Car type, Car name, Engine Num, Displacement, NumberCyl , Plate, Color, Mileage, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day	Car Rental System	
2. Car	Means-end Plan (Provide prices info)	*Composite* goal (Provide informa-tion	Car type, Car name, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day	Car Rental System	
3. Car	Resource Dependency (Info of the prices and models)		Car type, Car name, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day	User Company – Car Rental System	Divide
4. Extra services, Car, Extra services availability, Car avail-	*Composite* goal (Analyze avail-ability in this branch			Car Rental System	
5. Extra services, Car, Extra services availability, Car avail-	Means-end plan (Search car availability)	*Composite* goal (Analyze availabil-ity in this branch)		Car Rental System	
6. Extra services	Subplan AND (Obtain data Extra services)	*Composite* plan (Search car avail-ability)	Extra services type, Extra services name, model, Pickup Date, Return Date	Car Rental System	
7. Extra services	Resource Dependency (Data Extra services info)		Extra services type, Extra services name, Model, Pickup Date, Return Date	User Company – Car Rental System	
8.Car	Subplan AND (Obtain date and model car)	*Composite* plan (Search car avail-ability)	Car type, Pickup Date, Return Date, Pickup Zone	Car Rental System	

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
9. Car	Resource Dependency (Car wanted info)		Car type, Pickup Date, Return Date, Pickup Zone	User Company – Car Rental System	
10. Available Car	Subplan AND (Search car availability)	*Composite* plan (Search car avail-ability)	Car type, Pickup Date	Car Rental System	
11. Available Car	Resource Dependency (Car availability info)		Car type, Pickup Date, Pickup Zone	User Company – Car Rental System	
12. Available Car	Subplan AND (Give result the Car availability)	*Composite* plan (Search car avail-ability)	Car type, Pickup Date, Pickup Zone	Car Rental System	
13. Available Extra services	Subplan AND (Give result the Extra service availability)	*Composite* plan (Search car avail-ability)		Car Rental System	
14. Available Extra service	Resource Dependency (Extra service availability info)		Extra services type, Pickup Date	User Company – Car Rental System	
15. Available Car	*Composite* goal (Analyze availability in other branches)			Car Rental System	
16. Available Car	*Means-end* (Solicit availability in other branch)	*Composite* goal (Analyze availabil-ity in other	Car type, Pickup Date, Return Date, Branch	Car Rental System	

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
17. Available Car	Resource Dependency (Reservation info)		Car type, Pickup Date, Return Date, branch	Car Rental System – other branches	
18. Available Car	Means-end (Obtain info availability)	*Composite* goal (Analyze availability in other branches)	Car type, Pickup Date, Return Date, Branch	Car Rental System	
19. Available Car	Resource dependency (Availability info)		Car type, Pickup Date, Return Date, Branch	Car Rental System – other branches	
20. Customer, Person, Company manager, Company	*Composite* goal (Analyze customer)			Car Rental System	
21. Customer, Person, Company manager, Company	*Means-end* plan (Analyze the customer info)	*Composite* goal (Analyze customer)		Car Rental System	
22. Customer, Person, Company manager, Company	Subplan AND (Obtain customer info)	*Composite* plan (Analyze the customer info)	Same that concerned objects 23.	Car Rental System	

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
23. Customer, Person, Company manager, Company	Resource dependency (Customer info)		Name, Title, License Number, Birthday (Person) Agent name, Department, Passport Number, License, (Company manager) Agency Code, contact Name (Company)	User Company – Car Rental System	
24. Customer	Subplan AND (Search the Customer info)	Means-end plan (Analyze the customer info)	Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, birthday, N-Credit card, Card holder,	Car Rental System	
25. Customer	Subplan AND (Analyze Customer info)	Means-end plan (Analyze the Customer info)	Name, Passport Number, Address, City, Home phone, License Number, Status Customer	Car Rental System	
26. Reservation, Services, Car, Customer, Person, Company manager, Company, Credit card	*Composite* goal (Register reservation)			Car Rental System	

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
27. Reservation, Extra services, Car, Customer, Person, Company manager, Company, Credit card	Means-end plan (Register rent payment)			Car Rental System	
28. Customer	Subplan AND (Register Customer)	*Composite* plan (Register rent payment)	Name, Title, License Number, Birthday (Person) Agent name, Department, Passport Number, License, (Company manager) Agency Code, contact Name (Company)	Car Rental System	
29. Reservation	Subplan AND (Register reservation)	*Composite* plan (Register rent payment)		Car Rental System	
30. Reservation, Car	Resource dependency (Reservation info)		ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Type Car, Plate, Color, Mileage, Status, Customer type, Name, Customer passport	User Company – Car Rental System	
31. Extra services	Subplan AND (Register extras services)	*Composite* plan (Register rent payment)		Car Rental System	

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
32. Extra services, Car	Resource de-pendency (Extra services info)		Extra services, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Customer, Name, Customer passport	User Company – Car Rental System	
33. Car	Subplan AND (Register car as reserved)	*Composite* plan (Register rent payment)	Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Color, Mileage, Status, Customer type, Name, Passport Customer, Status car	Car Rental System	
34. Car, Extra service	Subplan AND (Register Extra services as re-served)	*Composite* plan (Register rent payment)	Extra services, Extra serviceNum , Pickup Zone, Re-turn Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Customer, Name, Customer passport, Status Extra service	Car Rental System	
35. Reservation other branch	Subplan AND (Register reserva-tion of the car in other branch)	*Composite* plan (Register rent payment)	ReservNum, Pickup Date, Return, Pick up-Hour, Re-turn Hour, Car type, Plate, Color, Mileage, Status, Branch	Car Rental System	
38. Credit card	Resource de-pendency (Valida-tion credit card)			Bank – Car Rental System	
39. Credit card	Resource de-pendency (Credit card info)		Customer, No. Credit card, Expiration	Bank – Car Rental System	

Concerned object name	Elements and associated links	Associated elements	Used attributes	Related actors	Label of the concerned object
40. Insurance, Car, Customer	*Composite* goal (Contract insurance)		Customer, Car type,	Car Rental System	
41. Insurance, Car	*Means-end* plan (Insurance vs. accident)	*Composite* goal (Contract insurance)		Car Rental System	
42. Insurance, Car, Customer	Resource dependency (info insurance)		Car type, plate, Customer type, Name, Customer passport, Pickup Date, Return, Pick up-Hour, Return Hour,	Insurance – Car Rental System	

Table 6.6 shows the table resulting from the application of the optimization criteria, where the concerned objects are joined based on the semantic proximity of the concerned attributes.

Table 6.6 Appropriate concerned objects by optimization criteria for the *Car Rental* case study

Concerned object name	Used attributes	Label of the Concerned object
Car	Car type, Car name, Engine Num, Displacement, NumberCyl , Plate, Color, Mileage, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day, Pickup Date, Return Date, Pickup Zone, Branch	Grouped Available Car
Extra services	Extra services type, Extra serviceNum, Extra services name, model, Pickup Date, Return Date	Grouped Available Extra service
Customer	Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, Birthday, N-Credit card, Card holder, expire data, Company	
Person	Name, Title, License Number, Birthday	
Company manager	Agent name, Department, Passport number, License	
Company	Agency Code, Contact Name	
Reservation	ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, plate, Color, Mileage, Status, Customer type, Customer name, Customer passport	Grouped Reservation other branch
Credit card	Customer, No. Credit card, Expiration	
Insurance	Car type, Plate, Customer type, Customer name, Customer passport, Pickup Date, Return, Pick up-Hour, Return Hour	

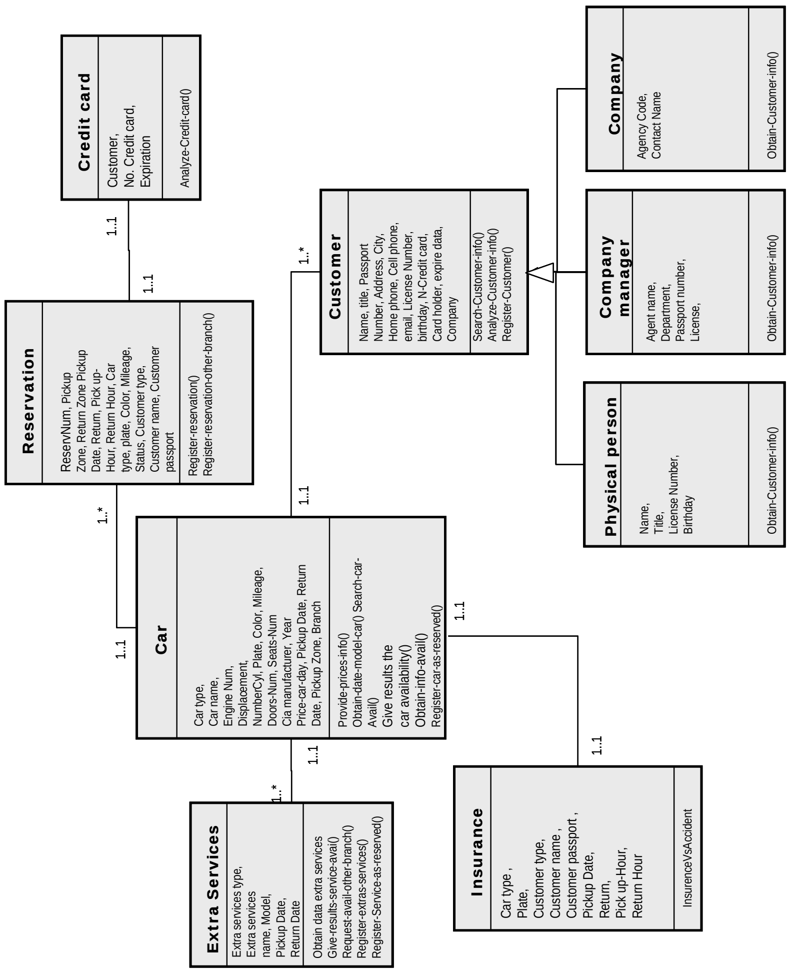
Table 6.7 shows the table resulting from the application of the modularity criteria, where the concerned objects are divided to create more specific objects based on the usability factors.

Table 6.7 Appropriate concerned objects by modularity criteria for the *Car Rental* case study

Concerned object name	Used attributes	Label of the Concerned object
Car	Car type, Name car, Engine Num, Displacement, NumberCyl, Plate, Color, Mileage, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day, Pickup Date, Return Date, Pickup Zone	
Available Car	Car type, Plate, Pickup Date, Return Date, Branch, Pickup Zone	
Engine	Engine Num, Displacement, NumberCyl	Divide Car
Extra services	Extra services type, Extra serviceNum, Extra services name, Model, Pickup Date, Return Date	
Available Extra service	Extra services type, Extra serviceNum, Pickup Date	
Customer	Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, birthday, N-Credit card, Card holder, expire data, Company	
Person	Name, Title, License Number, Birthday	
Company manager	Agent name, Department, Passport number, License	
Company	Agency Code, Contact Name	
Reservation	ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Color, Mileage, Status, Customer type, Customer name, Customer passport	
Reservation other branch	ReservNum, Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Color, Mileage, Status, Branch	
Credit card	Customer, No. Credit card, expiration	
Insurance	Car type, Plate, Customer type, Customer name, Customer passport, Pickup Date, Return, Pick up-Hour, Return Hour	

Table 6.6 and Table 6.7 are the input for the process of generation of the conceptual model. As can be analyzed from this table, the classes generated from the application of different optimization criteria have

significant differences that will have a correspondence in the generation of the conceptual schema. Figure 6.20 shows the conceptual model generated from the optimization criterion using the proposed transformation rules. Figure 6.21 shows the conceptual model generated from the modularity criterion.


Credit card
Customer, No. Credit card, Expiration
Analyze-Credit-card()
Company
Agency Code, Contact Name
Obtain-Customer-info()
Customer
Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, birthday, N-Credit card, Card holder, expire data, Company
Search-Customer-info() Analyze-Customer-info() Register-Customer()
Company manager
Agent name, Department, Passport number, License,
Obtain-Customer-info()
Reservation
ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, plate, Color, Mileage, Status, Customer type, Customer name, Customer passport
Register-reservation() Register-reservation-other-branch()
Physical person
Name, Title, License Number, Birthday
Obtain-Customer-info()
Car
Car type, Car name, Engine Num, Displacement, NumberCyl, Plate, Color, Mileage, Doors-Num, Seats-Num Ca manufacturer, Year Price-car-day, Pickup Date, Return Date, Pickup Zone, Branch
Provide-prices-info() Obtain-date-model-car() Search-car-Avail() Give results the car availability() Obtain-info-avail() Register-car-as-reserved()
Extra Services
Extra services type, Extra services name, Model, Pickup Date, Return Date
Obtain data extra services Give-results-service-avail() Request-avail-other-branch() Register-extras-services() Register-Service-as-reserved()
Insurance
Car type , Plate, Customer type, Customer name , Customer passport , Pickup Date, Return, Pick up-Hour, Return Hour
InsurenceVsAccident
1..1
1..1
1..*
1..*
1..1
1..1
1..1
1..1
1..*
1..1
Figure 6.20 Conceptual Model for the optimization criterion

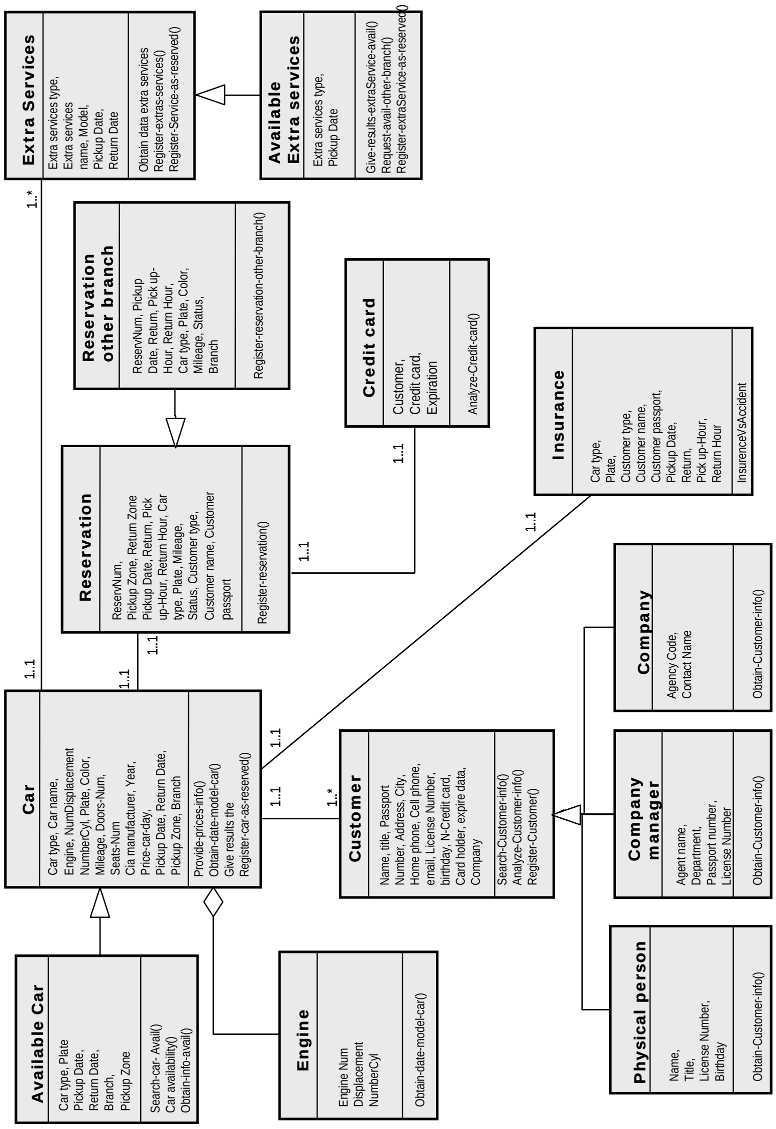

Extra Services
Extra services type, Extra services name, Model, Pickup Date, Return Date
Obtain data extra services
Register-extras-services()
Register-Service-as-reserved()
Available Extra services
Extra services type, Pickup Date
Give-results-extraService-avail()
Request-avail-other-branch()
Register-extraService-as-reserved()
Reservation other branch
ReservNum, Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Color, Mileage, Status, Branch
Register-reservation-other-branch()
Reservation
ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Mileage, Status, Customer type, Customer name, Customer passport
Register-reservation()
Credit card
Customer, Credit card, Expiration
Analyze-Credit-card()
Insurance
Car type, Plate, Customer type, Customer name, Customer passport, Pickup Date, Return, Pick up-Hour, Return Hour
InsurenceVsAccident
Car
Car type, Car name, Engine, NumDisplacement NumberCyl, Plate, Color, Mileage, Doors-Num, Seats-Num Cia manufacturer, Year, Price-car-day, Pickup Date, Return Date, Pickup Zone, Branch
Provide-prices-info()
Obtain-date-model-car()
Give results the
Register-car-as-reserved()
Customer
Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, birthday, N-Credit card, Card holder, expire data, Company
Search-Customer-info()
Analyze-Customer-info()
Register-Customer()
Company
Agency Code, Contact Name
Obtain-Customer-info()
Company manager
Agent name, Department, Passport number, License Number
Obtain-Customer-info()
Physical person
Name, Title, License Number, Birthday
Obtain-Customer-info()
Available Car
Car type, Plate Pickup Date, Return Date, Branch, Pickup Zone
Search-car- Avail()
Car availability()
Obtain-info-avail()
Engine
Engine Num Displacement NumberCyl
Obtain-date-model-car()
1..*
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..*

Figure 6.21 Conceptual Model for the modularity criterion

6.5 Dynamic Model Generation

The *ONME* dynamic model is represented by state transition diagrams defined for each one of the classes in the object model. This model represents the valid states in the lifetime of the objects of the software system.

The generation of the dynamic model is directly associated to the changes of values of the attributes. In this sense, a transition between states implies changes in the value of the object attributes as result of the application of a certain task. In the current proposal state, there are no indications about the values of the attributes in the organizational processes. The changes in the attributes values have not been considered in our proposal in order to avoid overloading the resultant model. However, if the dynamic model needs to be generated, the changes in the attributes must be considered.

To obtain this model, the lifetime of the concerned objects needs to be analyzed in order to detect the changes in the states and values of the concerned objects (which are used to generate the classes in the object model).

As stated above, we register all the occasions where a concerned object is created, manipulated or deleted in the concerned objects model. Therefore, this model is correctly adapted to determine the state transition diagram. Thus, a) when an object is created the first state of the object is created, b) when a modification of the attributes of the object is detected, this indicates a transition to the following state. c) If the value of the attributes back to previous values, then a transition towards a previous state must be indicated. It is important to point out that this is a preliminary analysis to obtain a dynamic model.

6.6 Functional Model Generation

The *ONME* functional model represents the changes in the values of the objects when a transition between states occurs. To obtain this model, the lifetime of the concerned objects needs to be analyzed in order to detect the changes in the states of the concerned objects. The concerned object model has the register of all changes to the concerned objects. As stated above, the analysis of the changes in the values of the attributes is a required condition to generate the dynamic and functional models. Thus, modifications to the current method must be proposed in order to consider this modeling element. The generation of the functional model complements the generation of the dynamic model, but in the generation of the functional model the analysis is focused on registering the changes in the value of the attributes of the concerned objects.

6.7 Summary

One of the main contributions of our research work is the definition of the concerned object model as an intermediate representation between the organizational model and software representation models. The concerned object model has the appropriate abstraction level to obtain the elements of an object-oriented conceptual model. This is because the concerned model contains information about the lifetime of the relevant objects in the organizational process. The semantic proximity between the concerned object model and the object-oriented conceptual schema enables the analyst to use the latter to obtain the following elements of the object model: classes (attributes, and methods), association and aggregation relationships, and triggers.
The proposed method enables the analyst to produce an alternative set of conceptual models according to a specific optimization criterion. This approach, which has been historically used in database

Chapter 7

Linking late requirements with the *ONME* requirements model

In this Chapter, we introduce the proposed method to generate an *ONME*[1] requirements specification for the system-to-be from the late requirements models defined in previous modeling steps. More specifically, the proposed method enables the analyst to generate functional requirements, represented as UML use case diagrams, from organizational models specified in the Tropos framework.

[1] Olivanova Model Execution

This Chapter is structured in two sections: In the first section, we briefly explain the foundations on requirements modeling. In the second section, the proposed method is presented in detail.

7.1 Introduction

Nowadays, there is a wide consensus with respect to considering requirements engineering as a fundamental activity in the process to design and develop a software product. Traditionally, requirements engineering has been defined as the systematic process of identification and specification of the expected functions of a software system.

Some authors have stated the definition of requirements engineering as *"the science and discipline concerned with analyzing and documenting requirements"* [Dorf90]. Kotonya restated the definition of requirements engineering as *"the systematic process of eliciting, understanding, analyzing, documenting (and managing) requirements"* [Koto98]. Alexander [Alex02] and Hull [Hull02] discuss additional properties of the text-based requirements (e.g. priority and traceability) in conjunction with guidelines to improve writing of requirements. The IEEE Computer Society [IEEE03] states *"Requirements identify the purpose of a system and the contexts in which it will be used. Requirements act as the bridge between the real world needs of users, customers and other stakeholders affected by the system and the capabilities and opportunities afforded by software and computing technologies. The construction of requirements includes an analysis of the feasibility of the desired system, elicitation and analysis of stakeholders' needs, the creation of a precise description of what the system should and should not do along with any constraints on its operation and implementation, and the validation of this description or specification by the stakeholders. These requirements must then be managed to consistently evolve with the resulting system during its lifetime."*

In the software community, it is clear that when a software product is designed and implemented, it is very important to ensure from the beginning that the user requirements have been properly represented.
In recent years, many research efforts have been made to define software production processes to generate software system from software requirements. Some of these works offer a precise, rigorous and reliable production process [Serd91] [Past99] where the system is the result of the correspondence among the elements of a software requirements specification and the implementation elements in a target language. Most of these works use system requirements (late requirements) as a starting point to develop the system. Even if this approach solves many of the problems associated with the generation of the software product, it does not ensure that the system integrates the functionalities expected by the organizational users.
In these production processes, there is one main feature that is not properly taken into account: the importance of understanding that the information system should be the correct representation of the requirements taken from the organizational model. McDermind [McDe94] indicates that when the functional specification of the software system is the focal point of the requirements analysis, requirements engineers tend to establish the scope of the software system before having a clear understanding of the user's real needs. It constitutes a very important reason why many of the systems developed from a requirements model that focus only on the functionality of the software system do not comply with their correct role within the organization.
In a software production process that does not have the organizational process model as a first stage, any attempt to generate a prototype of the information system will be reduced by the incapacity to assure beforehand the real usefulness of the system in the context of the organizational tasks. We consider that it is only possible to generate a software system that complies with the users´ needs if the software engineers have a precise knowledge of the way in which the organization works.
There are research works that highlight the importance of using organizational models as a starting point in the development of

information systems [Bube94] [Cesa02] [Louc95] [Cast02]. However, there is currently no software development environment that offers a methodological approach that is based on a organizational model for the generation of prototypes of information systems.

One of the reasons for the lack of methodological solutions to incorporate organizational modeling (early requirements) as a key requirements engineering process is the difference that exists between the abstraction levels of the two specifications. The lack of traceability methods has affected the practical application of the organizational model technique in integrated software production process environments. Thus, we argue that the determination of a methodological approach to use the elements of an organizational model to obtain the expected functionalities of the information system is a basic requirement to assure its usefulness in practice.

In this Chapter, we present a method to generate information system requirements from an organizational model represented in the Tropos Framework. The requirements specification (use cases and scenarios) generated corresponds with a specific requirements approach RETO[1], which is the requirements method and tool associated to the OO-Method CASE Tool. By doing this, we take a further step in the process of integrating organizational modeling into industrial software production process.

[1] Requirements Engineering TOol

7.2 Foundations of the requirements model the *ONME* requirements model

This thesis has been developed within the context of the OO-Method project, which is an object-oriented method that provides a set of well-defined and complementary graphical techniques to build a conceptual schema and requirements model of the system-to-be. OO-Method has an industry-oriented implementation called OlivaNova Model Execution (*ONME*) [Oliv07], which is a CASE Tool that provides an operational environment that supports all the methodological aspects of the OO-Method which has been developed in the context of an R&D project carried out jointly by the Valencia University of Technology, CARE Technologies SA and Consoft SA in Spain.

Nowadays, the *ONME* starts the software production process with the definition of the late requirements for the system-to-be. One of the aims of this thesis is to add an early requirements phase to the *ONME* production process. To do this, the proposed method must connect its resultant models with RETO, the software requirements engineering tool of the *ONME*.

RETO defines a requirements model, which captures both functional and usage aspects in a comprehensive manner. This is organized through the use of three complementary techniques: the mission statement, the function refinement tree and the use case diagrams.

The techniques that are used in RETO Requirements method include:

The ***Mission statemen***t: it describes the purpose of the system in one or two sentences. It also describes the major responsibilities as well as a list of things the system is not to do. External interactions can always be partitioned into functions. It is very useful to organize these functions in a refinement hierarchy so that the root of the hierarchy is the overall system function (the mission statement), and the leaves are the elementary functions. The intermediate nodes are groups of elementary functions and usually represent a kind of activity or an area of business where the system is under development.

The ***Function refinement tree***: it deals with external interaction partitioning according to the different business areas or business objectives. The function refinement tree can be used to represent a hierarchical decomposition of the business functions of a system which is independent from the current system structure. The resultant tree is merely an organization of external functions and does not say anything about the internal decomposition of the system. However, it gives the entry point for building the use case model instead of starting from scratch and avoids the potential problem of mixing the abstraction level of use cases.

The ***Use case model***: it includes the use case specification to specify the composition of external interactions and the use case diagram to show communication between the environment (actors) and the system.

The use of the mission statement and the function refinement tree together with the use case model is the key to finding a good abstraction level for use cases that answer the question of what a use case really is.

RETO gives methodological guidance to convert these requirements into a precise conceptual schema (provided by the OO-Method conceptual modeling constructs); RETO then links this conceptual schema with the model-based code generation techniques of the OO-Method in order to automatically generate the software system.

7.2.1 Requirements models

The purpose of the Requirements Model is to understand what is to be built and to provide techniques to accurately capture the desired properties for it. Furthermore, the purpose is to build a model of these requirements in a manner that people without formal training in the notation can understand and review.

One of the more influential techniques in software requirements are scenarios. The scenario-based techniques have been used in software engineering to understand, model and validate user requirements in a non-formal manner. Some of the most relevant scenario-based techniques are [Haum98] [Roll98] [Leit97] [Jaco95b]. These research works use scenarios to elicit and validate requirements.

Among the scenario-based techniques, use cases have been receiving special attention in the software engineering community. The use case modeling introduced in UML (Unified Language Modeling) by Jacobson et al [Jaco92] is currently considered to be one of the most relevant tools for capturing system requirements. Use cases capture the system as it is viewed from the outside and depict the interaction between the system and external actors. A use case describes the sequence of steps that is performed by a user who interacts with a system to accomplish a task or goal. However, the description of use case only concerns what system functionalities exist, not the details of the implementation of these functionalities. Use cases have become one of the most popular techniques in object-oriented methods. Even though they are described in an informal technique; the use cases are widely used and have obtained a central place in system development [Fowl98]. Use cases are valuable for several reasons. First, they help discover requirements. Use cases allow you to capture a user's need by focusing on a task that the user needs to do. Use cases can also help formulate system tests to ascertain that the use case is indeed built into the system.

Since use case diagrams provide a clear way to represent the structure of the requirements in a software system, they easily serve as a means of communication between software developers and users.

A use case diagram consists of a set of use cases, actors and their relationships with each other. A use case represents a functionality of the system-to-be, i.e, what the system does. An actor is an outside user of the system, and the actor can interact with use cases defined in the system. The relationships in a use case diagram can be divided into four categories: *association, include, extend* and *generalization*. However, when describing software requirements, in most cases, the four categories are not enough to represent the complete functionality of the system-to-be. Therefore, some descriptions for each use case such as main flow of events, precondition, post condition and exceptional flow events should be provided as supplements to a use case diagram. All of this graphical and textual information yields a complete requirements model for a software system.

7.3 The generation process of the requirements model

This section describes our method for generating a requirements model from organizational models presented in the Tropos Framework.
The generation of a requirements model, which is represented using UML use case models, is the result of a deep analysis of the organizational context. Therefore, the analysis presented in previous chapters (analysis of organizational goals, extension of the organizational model with the software system actor and extension of the organizational model with the concerned objects) are the basis to provide the appropriate information to generate the requirements model.
Santander and Castro [Sant01] have studied the generation of the use case models from organizational models. Their approach focuses on the translation process of the organizational models into a use case model specification. To do this, the elements of the organizational model are directly associated to the elements of the textual scenarios of a use case model. However, the main issue of this work is the lack of an intermediate step that allows the analyst to filter the relevant information to be considered in the definition of the system-to-be. Also, there are no guidelines in this work to help the analysts the model generation. Therefore, it is complicated to generate a organizational model that is correctly adapted to generate requirements specifications. In our research work, we put more emphasis on organizational model creation by providing guidelines that allow us to generate organizational models adapted for use case generation.
Following, a brief overview of the method for obtaining requirements is presented. The *Car Rental* case study is used to illustrate the proposed method.

Overview of the proposed method

One of the main objectives of this thesis is to define a systematic approach to generate late requirements specifications that correctly fit the objectives of the organizational actors.

The process begins with the understanding of the business context. To do this, a goal analysis method must be performed to determine the business objectives and the alternative solutions given to fulfill these goals (Chapter 3). As a result of this process, the plans to be automated are identified (Chapter 3); afterwards the software system actor (SSA) is inserted into the organizational model and the relevant organizational elements to be automated through the system-to-be are delegated to this new actor. This process permits the abstraction level of the organizational model to be reduced so that it is closer to software specifications (Chapter 4). The next process consists of inserting the monitoring plans and extending organizational model with the relevant objects, which we called concerned objects (Chapter 5). The resultant model of these modeling stages is used as input for the generation process of the requirements model explained in this Chapter (Figure 7.1).

The proposed generation approach is composed of two steps that guide the process of mapping between the organizational models and the use case model specified in UML. This is done by defining the correspondence between the elements of the organizational model and the use case model and its corresponding scenarios. The first step is the generation of functional groups. In this step, the functionality of the software to-be is organized in packages. In the second step, the process to discover the use case model is carried out by doing the following:

- Discovering default use cases
- Discovering use cases through the analysis of the SSA
- Discovering use case actors
- Discovering relationships between use cases
- Building scenarios for use cases

The generated model will contain a package diagram with all the identified functional groups and the use case diagrams and their scenarios.

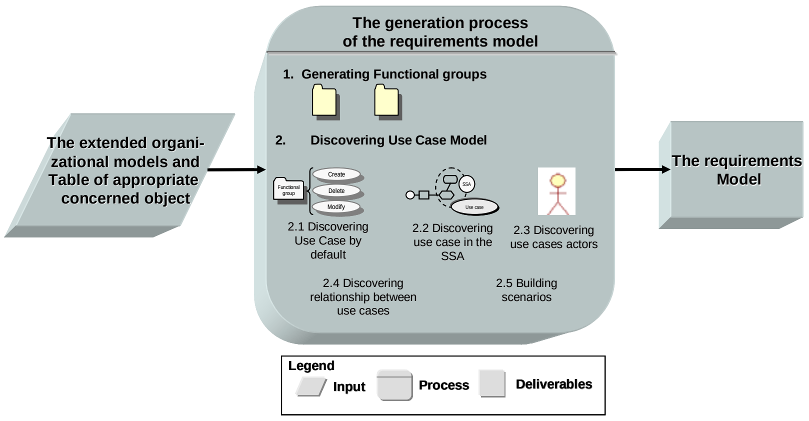

The extended organi-zational models and Table of appropriate concerned object
The generation process of the requirements model
1. Generating Functional groups
2. Discovering Use Case Model
Functional group
Create
Delete
Modify
SSA
Use case
2.1 Discovering Use Case by default
2.2 Discovering use case in the SSA
2.3 Discovering use cases actors
2.4 Discovering relationship between use cases
2.5 Building scenarios
The requirements Model
Legend
Input
Process
Deliverables

Figure 7.1 Overview of the Requirements Engineering Generation Method

The use case model obtained from the application of the proposed approach will be the source model for the RETO requirements tool. It is important to point out that not all the information of the use case model can be generated from the organizational model in an automatic manner. This is because some of the sections of a use case specification (i.e., the building scenarios detailed in section 7.3.3) are the result of abstraction mechanisms of the software specification.

7.3.1 Generating functional groups

The first step of the generation process of the requirements model is the generation of functional groups. A functional group describes the different subsystems that an information system can be divided into. Each functional group makes reference to an element that is manipulated (through user's interactions) by the software system. In UML, this modeling technique is called package diagram and can be used to group objects that provide related services. The package has responsibilities that are strongly related. The package has low coupling and low cohesion with respect to interfacing with other packages in the system [Larm01].

In this proposal, the functional groups are used to classify the functionalities of the software system to-be. Each functional group includes the whole information about the resource being manipulated, generated or obtained by the system in an automatic way.
The graphical presentation of the functional group is a tabbed folder, where the name of the functional group must be written on the tab or inside the folder. Figure 7.2 depicts this primitive.

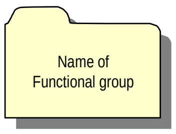


Figure 7.2 Graphical representation of functional group

Defining functional groups
In our approach, the model source to obtain the functional groups is the actor diagram that has been extended with the concerned objects. In this model, the dependencies that associate a organizational actor and the SSA will generate the functional groups. To do this, Rule 1 and Rule 2 need to be applied.

> **Rule 1.** Each concerned object identified in a resource or plan dependency between an organizational actor and SSA must be mapped in a functional group.
>
> The reason for doing this is that a dependency between an actor and the SSA implies an interaction between the actor and the software system to manage a specific organizational resource. Thus, this resource needs to be created and manipulated by the system, which implies the creation of use cases (contained in the functional group) to manipulate this informational resource. Nevertheless, before creating a functional group the duplicity of the functional group in the requirements model must be verified. This is because the functional group could be created in another dependency relationship with the same concerned object. Figure 7.3 illustrates an example of the creation of a functional group in the extended actor diagram.

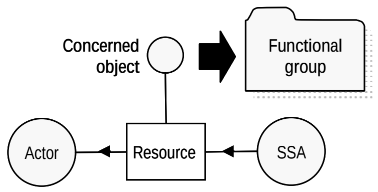


Figure 7.3 Creation of the functional groups in the extended actor diagram

With regard to the *Car Rental* case study, Figure 7.4 shows the concerned object *Customer* identified in the resource dependency *Customer info* between the *clerk* actor and the SSA. This concerned object is mapped in a functional group.

Rule 2. The name of the functional group is composed of the name of the concerned object and the word "*Management*". If the concerned object is manipulated in several situations, then the name must be written in plural. Following with the example of Figure 7.4, the functional group has the same name as the concerned object.

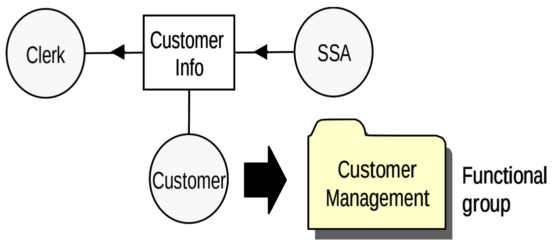


Figure 7.4 Customer Management functional group

7.3.2 Discovering the use case model

The second step in constructing the late requirements model for the system-to-be is the generation of the use cases that are associated to the functional groups. In following sections, each sub-step is explained in detail to show how the use case model from the organizational model is determined. Also, one or more scenarios must be provided for each use case to expresses how the system should interact with the end user or another system to achieve a specific organizational goal.

7.3.2.1 Discovering use cases by default for each functional group

During the development of case studies for this thesis, we found that a set of basic use cases must be defined in each functional group to manage the analyzed informational resource (create, delete and modify elements). These default use cases must be inserted in each functional group to ensure the correct management of the analyzed requirements. Rule 3 defines the creation of these default use cases.

> **Rule 3**. Default use cases *Create, Delete* and *Modify* must be created for each functional group elicited in the previous steps (Figure 7.5). These use cases allow us to ensure the appropriate management of each functional group.

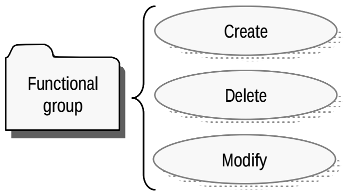


Figure 7.5 Use cases created by default

Figure 7.6 illustrates the functional group *Customers Management* for the running example. The use cases created by default are: *Create Customers, Delete Customers and Modify Customer*s.

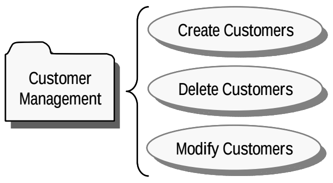


Figure 7.6 Example of use cases created by default for the Customer Management functional group.

7.3.2.2 Discovering use cases in the SSA

Once the default use cases have been created for the elicited functional groups, the next step consists of determining the use cases from the organizational model that was extended with the inclusion of the software system actor. Therefore, an analysis of the internal element of the SSA must be carried out to determine its relevance in defining use cases for the system-to-be. As mentioned above, the internal elements that compose the software system actor (SSA) are goals and plans, which are associated through means-end and decomposition links to compose a tree structure that reflects the functions that must integrate the system-to-be. Therefore, the SSA integrates one or more tree structures that correspond to goals that have been delegated from the organizational actors.

The strategy to discover candidate use cases from the internal actors in the SSA consists in traversing the tree structures looking for evidence that an internal element can address a use case. This process is organized in three complementary steps.

Step 1. An in-order traversing must be carried out to analyze each internal element of the SSA. This procedure is similar to the one illustrated in the second step of the method to apply the proposed pattern language (Chapter 4, section 4.4.3) to delegate plans and goals towards the SSA.

Step 2. For each internal element analyzed in the exhaustive tree traversing, it is necessary to determine if it can be considered as a candidate use case. If an internal plan or goal is involved in a dependency relationship, then this element is a candidate for a uses case. A dependency relationship implies an explicit interaction between the SSA and organizational actors that uses the system to perform a specific functionality. This approach fits the standard concept of use cases, which considers that use cases are interactions between a user and the system-to-be in order to achieve a goal [Cock01].

In this proposal, if an internal plan or goal is not involved in a dependency relationship, then this element is a candidate to be part of a use case (i.e., as a step in the scenario that describes a use case, or it can be a candidate to create a specific use case that is included for a general use case). The rules associated with this step are the following:

Rule 4. Each plan within the SSA that is directly involved in a dependency relationship will be a candidate to be a use case in the requirements model. For example, Figure 7.7 shows the plan *Obtain Customer info*, which is involved in a dependency relationship between the SSA and the Clerk actor. Therefore, it can be considered as a candidate to be a use case.

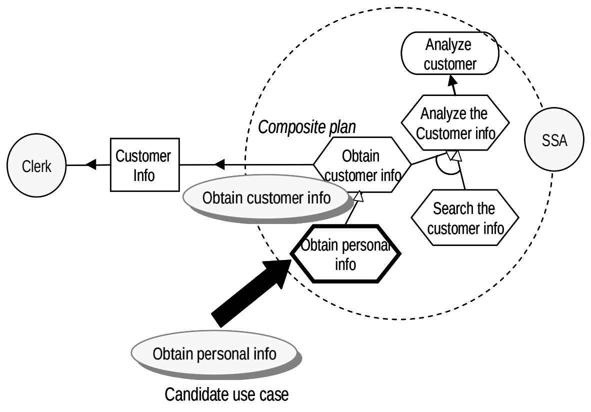


Figure 7.7 Example of a use case generated from an internal plan.

Rule 5. Each plan within the SSA that is not involved in a dependency relationship could be a candidate to be a part of another use case in the requirements model. If the analyzed plan has a *composite* plan[1] as a parent node, which, in turn, is involved in a dependency relationship, then the analyzed plan will be a use case that is linked to the generated use case for its node parent. These use cases will be linked through an «include» relationship.

Figure 7.7 shows an example of Rule 5. In this example, the plan *Obtain personal info* is the child node of the *composite* plan *Obtain Customer info*, which is involved in the resource dependency *Customer info*. This configuration creates a use case

[1] A *composite plan* is a plan whose execution is carried out by decomposing it into other sub-plans.

manage the Customer (obtained from the *composite* plan *Obtain Customer info* using Rule 4). A use case is also created from the plan *Obtain personal info* using Rule 5.

Rule 6. Each monitoring plan inserted in the SSA will be a candidate to be a use case in the requirements model. For example Figure 7.8 shows the monitoring plan *Analyze availability of cars*, which does not have a dependency relationship between the SSA and another organizational actor; however, it can be considered as a candidate to be a use case.

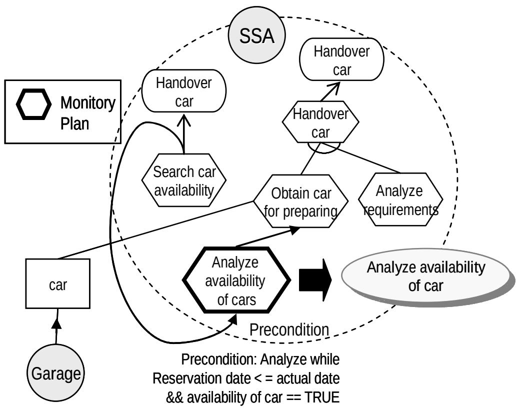

SSA
Monitory Plan
Handover car
Handover car
Handover car
Search car availability
Obtain car for preparing
Analyze requirements
car
Analyze availability of cars
Analyze availability of car
Precondition
Garage
Precondition: Analyze while Reservation date < = actual date && availability of car == TRUE

Figure 7.8 Example of a use case generated by a monitoring plan

Step 3. This step consists of determining the functional group for each use case identified in the previous steps. The rule associated to this step is the following:

Rule 7. If the plan or goal (which has generated a use case) is linked to a dependency relationship, then it is necessary to determine if this use case must be contained in the functional group created from the dependency relationship. Before allocating the use case in the functional group, it is necessary to analyze if the candidate use case corresponds with the semantics of some use case created by default in the functional group (*Create*, *Destroy* or *Modify*). If so, the candidate use case must substitute the use case created by default. Figure 7.9 illustrates an example of the application of Rule 7.

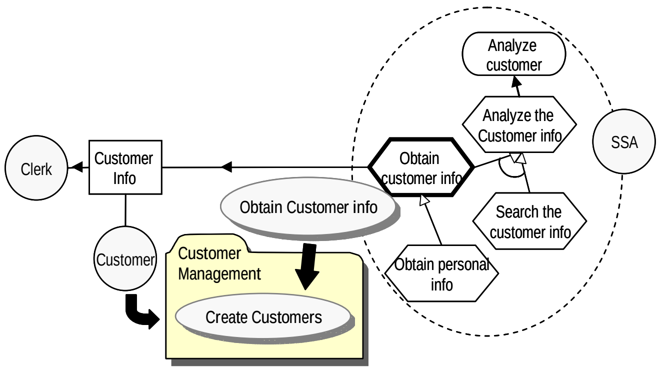


Figure 7.9 Example for determining a functional group for a use case

The use case generated from plan *Obtain Customer info* was included in the functional group *Customers Management*. This use case is used to carry out the register of the *Customers*; therefore, the use case *Obtain Customer info* must substitute the *Create Customers* use case which was created previously by default, when the functional group was identified. This process requires the intervention of the analyst to determine if some of the generated use cases must substitute the default use cases (*creation*, *deleted* or *modification* use cases)

7.3.2.3 Discovering use case actors

The third step to obtain the use case model of the system-to-be consists of discovering the use case actors. A use case defines a goal-oriented set of interactions between external actors and the system under consideration. *Actors* are parties outside the system that interact with the system [UML99]. An actor may be a class of users, roles that users can play, or other systems. Cockburn distinguishes between primary and secondary actors. A *primary* actor is one having a goal requiring the assistance of the system. A *secondary* actor is one from which the system needs assistance in order to satisfy its goal [Cock97].

In UML [UML07], an actor (of use cases) specifies a role played by a user or any other system that interacts with the subject. (The term "role" is used informally here and does not necessarily imply the technical definition of that term found elsewhere in this specification.). The notation of an actor is represented by "stick man" icon with the name of the actor in the vicinity (usually above or below) the icon.

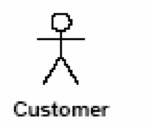


Figure 7.10 Notation of an actor in UML

In this proposal, the identification of actors is carried out by analyzing organizational actors and the roles or agents in the business, which have some kind of interaction with the SSA. Rule 8 defines the actor generation process.

Rule 8. The organizational actors with a dependency relationship with the SSA will be candidates to be actors of the requirements model. As commented above, a dependency relationship of a organizational actor with the SSA implies an explicit interaction that addresses a use case.

Rule 9. Plans without a direct association to dependency relationships do not give rise to actors. This is because these elements generate *included* use cases (which do not have a primary actor) or generate parts of the fragment of a use case, which already contains a primary use case actor.

Figure 7.11 shows an example of the application of Rule 8 to discover an actor of a use case. The plan *Obtain Customer info* has *g*enerated a use case with the same name as the plan. The dependency relationship associated to the plan is analyzed to determine the actor that participates in the dependency (*Clerk*). As a result of applying Rule 8, this organizational actor is translated into the actor that activates the use case *Obtain Customer info*.

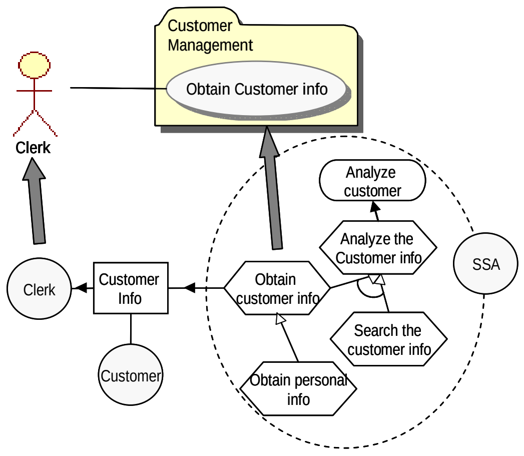


Figure 7.11 Example for discovering an actor of a use case

7.3.2.4 Discovering relationships between use cases

The fourth step of the process to generate the use case model consists in discovering the relationships between use cases. The UML standard supports three major relationships among use cases: include, extend and generalization; they can be summarized as follows [UML07]:

- **Include:**

  An include relationship between use cases specifies that an including (base) use case requires the behavior from another use case (the included use case). In an include relationship, a use case must use the included use case.

  The notation for the include relationships are denoted as dashed lines or paths with an open arrow-head pointing at the inclusion use case and are labeled with the «include» keyword (stereotype). The inclusion of a use case involves the execution of the base use case up to the inclusion point, inserting and executing the inclusion use case, and then continuing with the execution of the base use case. Figure 7.12 shows an example where use case *A* includes use case *B*.

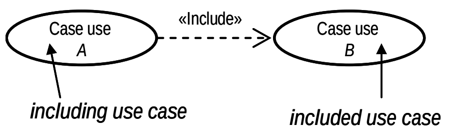


Figure 7.12 Example of the include relationship

In our method, the identification of «include» relationships between use cases can be carried out by analyzing the set of use cases generated through composition plan relationships. Rule 9 defines the generation of «include» relationships.

Rule 10. An *«include»* relationship must be created between use cases when the *composite* plan in a composition plan relationship has generated a use case and its associated subplans have also generated use cases (applying Rule 5). Therefore, an «include» relationship between these use cases must be created, where the use cases generated from *subplans* are included in the use case generated from the *composite* plan.

Figure 7.13 illustrates a partial view of the *Car Rental* case study. In this example, the application of Rule 4 to the *composite* plan *Obtain Customer info* generates a use case. A use case was also generated for the child node *Obtain personal info* through the application of Rule 5. Therefore, an *«include»* relationship between these use cases is created.

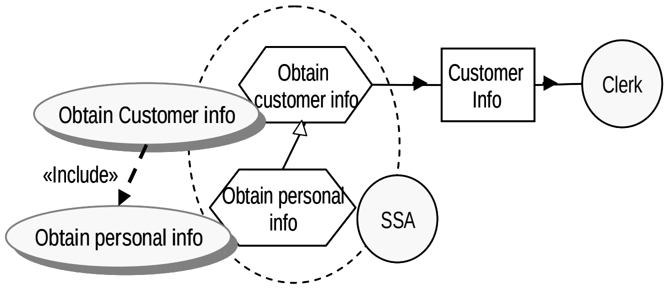


Figure 7.13 Example of the «include» relationship in the *Car Rental* case study

- **Extend:**

  It is a relationship from an extending use case to an extended use case that specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the extended use case [UML07]. The extension takes place at one or more specific extension points defined in the extended use case. Note, however, that the extended use case is defined independently of the extending use case and is meaningful independently of the extending use case. On the other hand, the extending use case typically defines behavior that may not necessarily be meaningful by itself. Instead, the extending use case defines a set of modular behavior increments that augment an execution of the extended use case under specific conditions.

  The notation for the extend relationship is shown by a dashed arrow with an open arrowhead from the use case providing the extension to the base use case. The arrow is labeled with the keyword «extend». The conditions of the relationship as well as the references to the extension points are optionally shown in a note attached to the corresponding extend relationship (See Figure 7.14).

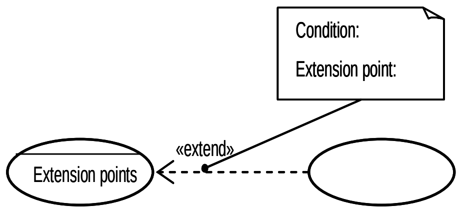
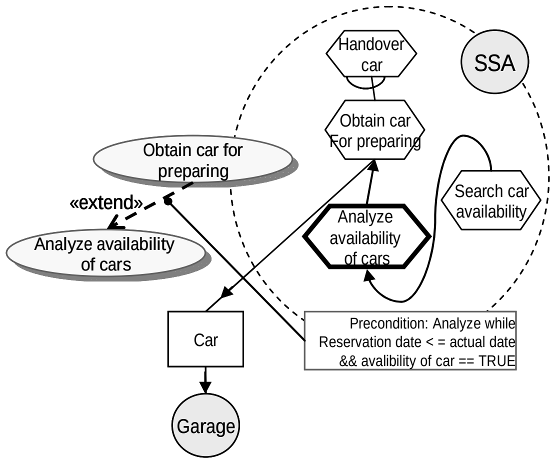


Figure 7.14 example of the extend relationship

Rule 11. An «extend» relationship must be placed between two use cases when a plan has been associated with a monitoring plan. Thus, this plan generates a use case and the monitoring plan also generates a use case.

In Figure 7.15, the plan *Obtain car for preparing* to be rented generates a use case (applying the Rule 4) and the monitoring plan *Analyze availability of cars* generates also a use case (applying the Rule 6). Therefore, an «extend» relation between these use cases must be created, where the precondition of this relationship will be the same than the precondition of the monitoring plan.



Handover car
SSA
Obtain car For preparing
Obtain car for preparing
«extend»
Search car availability
Analyze availability of cars
Analyze availability of cars
Car
Precondition: Analyze while Reservation date < = actual date && avalibility of car == TRUE
Garage

Figure 7.15 Example of the «extend» relationship in the *Car Rental* case study

- **Generalization:**

A generalization is a taxonomic relationship between a general classifier and a specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the general classifier.

A Generalization is shown as a line with a hollow triangle as an arrowhead between the symbols representing the involved classifiers.

The arrowhead points to the symbol representing the general classifier. This notation is referred to as the *"separate target style."* In this proposal, the identification of generalization relationships can be performed by analyzing the roles played between the organizational actors. Rule 12 details how to discover relationship of this kind in the requirements model.

Rule 12. The Tropos framework includes a modeling primitive to represent the roles that are played by the organizational actors. In our proposal, these roles are used to generate the generalization relationships.

To illustrate this situation, in Figure 7.16 we show an example of a generalization relationship among actors in the requirements model.

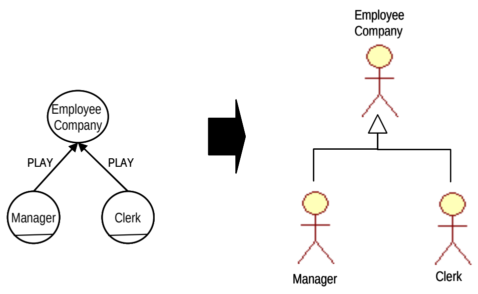


Figure 7.16 Example of generalization relationships

7.3.3 Building scenarios

The use case diagram visualizes the system's interactions and captures the scope of the system-to-be. The simplicity of these diagrams makes them a great communication tool. The use case itself details what the system must do, while the scenarios detail how the expected functionality must be performed by the system.

The last step of the method to discover the use case model consists in defining the scenarios of the use cases in accordance with RETO, the requirements engineering tool of the OO-Method approach [Insf03]. RETO uses the type of scenarios [Jaco92] [Hsia94] where entity types, not individual entities are used. Thus, they do not refer to *Smith* but to *customers*. Each execution of a type scenario is an instance scenario (or Use-Case instance).

RETO implements mechanisms to generate an object-oriented conceptual schema from the requirements model specification. The OO-Method, in turn, generates the fully functional software system from the conceptual model. In this thesis, our objective is to generate the RETO requirements model (use cases and scenarios) in order to provide a software development process that starts with organizational modeling activities and finishes with a fully functional software product.
The structure of the RETO use case description is composed of three sections: the first section is a *summary* of what the use case is about; the second section describes the *basic course of action*, which is the most important course of events, giving the best understanding of the use case. Variants of the basic course of events are represented in a third section, called the *alternative* section. A specified condition permits deciding which alternative to execute, and the flow of control is then transferred to this alternative.
An overview of the RETO structure for the use case scenarios is presented below.

1. *Use case summary section. This section contains information about the scenario. The information represented in the section is the following:*
    a) **Name**. This is the use case name.
    b) **Actors**. These are external agents that communicate with the use case, indicating who initiates the use case and the type of communication involved: input, output or input/output.
    c) **Pre-Condition**. This is a condition that should be satisfied in order to execute the use case
    d) **Purpose**. This is an explanation in natural language of what the use case is for.
    e) **Includes to**. This is a list of all individual use cases that are included in the use case model.
    f) **Extend to**. This is a list of all the use cases from which the current use case is an extension. For each use case, the corresponding *condition* and *extension point* in the referred use case is indicated.

2. *Basic course section. This section contains the steps that occur during the use case. This specification should include all the steps of the use case from the triggering event to the accomplishment of the goal. They are numbered paragraphs (steps) and are usually written in a conversational style between actors and the system [Larm01].*
3. *Alternative section. This section contains steps that complement the specification of the basic course of the scenario. The steps included in this section are not a course of steps on their own; they are only used in those cases where a given condition holds and some steps must be accomplished in order to complete the scenario. After the execution of this step, the original scenario continues its course of steps (e.g. in a car rental system, when a Customer returns a car, if the return hour is before the return hour of the contract, a discount is created for his or her personal account).*

An overview of this structure is illustrated in Table 7.1.

Table 7.1 Use case description structure

Name:			
Actors			
Purpose:			
Pre-condition:			
Includes to:			
Extends to:	UC Name	Extensions point	Condition

Basic Course Sections		
General	Actor/system communications	System response
Alternative Section		
Name:	Condition:	
General	Actor/system communications	System response

In this proposal, the process to build scenarios begins by selecting one of the elicited use cases. Then, the plan which was the source for that use case must be analyzed in order to obtain the use case scenario. In this case, the resource relationships associated to the plan that gives rise a use case are also a correct source for the generation of use case scenarios. The following rule helps in the construction of scenarios.

Rule 13. The resource relationships permit the functional groups to be determined; they also help to deduce the steps of the scenario.

For example, Figure 7.17 shows the resource dependency *Customer info*, where the SSA depends of the *Clerk* actor for obtaining the *Customer* information (i.e., N*ame*, *Passport-Number*, *Address*, *City*, *Home-home*, *License*, *Birthday*). In this way, some steps for the *Create Customer* use case can be deduced.

For example, (1) the system requests the *Customer* information, (2) the clerk introduces the *Customer* information, etc.

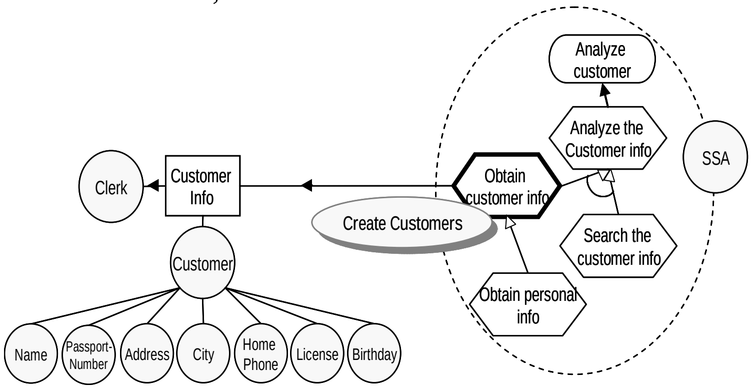


Figure 7.17 Resource relationship for obtaining some steps for the create *Customer* scenario

7.3.3.1 Guidelines to obtain use case scenarios from organizational models

The following guidelines were developed to help the analyst in the process of obtaining use case scenarios from organizational models.
Use case name**:** The use case name in the template of the scenarios will be the same as the use case elicited using Rule 3, Rule 4, Rule 5, and Rule 7, where the use cases were determined.
Use case actors**:** The actor (s) of the use cases will be those actors that interact with the SSA through dependency relationships which were source of use cases (Rule 8 and Rule 9).
Use case pre-Conditions**:** The preconditions for the use case will be the same as the precondition of the plan which generates the use case. As mentioned above, the formal definition, of the Tropos modeling elements include some aspects that were not represented in the graphical representation of the modeling diagrams. Two of these elements that are represented in the formal definition are the pre- and post-conditions of the organizational plans.
Use case purpose**:** The explanation about the purpose of the use case must be written by software engineers based on the goals that make operational the plans used to generate the use case model. These goals represent the rationalities behind the plans of the SSA.
Use case relationships**:** The relationships *include* and *extend* must be specified according to the relationships generated among internal plans in the SSA (Rule, Rule and Rule 12).
Use case basic course of action**:** This information will be obtained by analyzing the elements related to the plan that were the source of the generated use case. It is important to point out that decomposition and means-end relationships permit the analyst to detail the set of steps needed to fulfill a goal (means-end links) as well as the sub-steps required to perform an organizational plan (decomposition links). In the case of decomposition, the fulfillment of the child nodes implies the fulfillment of the parent node. Therefore, these internal refinement structures will be the basis to define the actions associated with a use case.
At this point, it is important to identify those resources or plans where the actors that are associated to the SSA play the role of

dependee in the dependency relationship (i.e., the system waits for actions or resources of the organizational actor) because the actions associated to this dependency must be used to indicate the user intervention (column *actor communications* in the scenario template). On the other hand, those resources or plans where the actors that are associated to the SSA play the role of *depender* in the dependency relationship (i.e., the organizational actors wait for actions or resources of the system) must be analyzed to specify the system responsibilities (column *system response* in the scenario template). Rule must be used to specify this situation.

***Alternative section*:** This section requires an in-depth analysis of the use case to determine whether the condition holds and to establish some steps to accomplish the entire scenario. Thus, this information must be completed by requirements engineers.

Example of the use case model of the *Car Rental* case study

An example of the use case model generated by applying our proposed rules to the running examples is shown in Figure 7.18, where the use cases of the *Customer* Management, Reservations Management, and Cars Management functional groups are shown. The use case description structure is shown in Table 7.2.

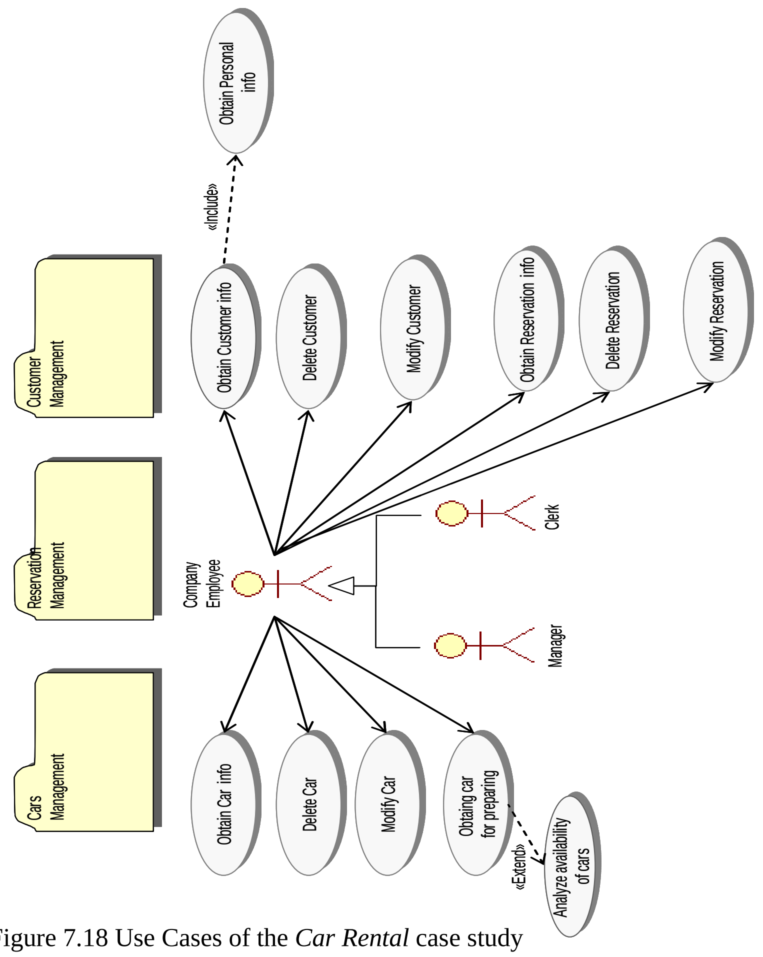


Figure 7.18 Use Cases of the *Car Rental* case study

Table 7.2 Example of the use case description structure

Name:	Create Customers		
Actors	Clerk		
Purpose:	The use case permits the information of a *Customer* to be register		
Pre-condition:			
Includes to:	Obtain personal info		
Extends to:	UC Name	Extension point	Condition

Basic Course Section		
General	Actor/system communications	System response
(1)The *Customer* wants to make a reservation		
	2) the Clerk selects the option of register a *Customer*	
		(3) the system requests *Customer* information
	(4) the Clerk introduces *Customer* information	
		(5) the system registers the *Customer*

7.4 Summary

One of the contributions of this thesis is a methodological approach to generate software requirements from organizational models.
A set of guidelines has been developed to establish the correspondence between the modeling elements of an organizational model and the modeling elements of a requirements model for the system-to-be. The guidelines help the analyst to define the system functionalities from the organizational plans. To do this, several steps must be fulfilled to generate an organizational model that integrates the software system as an explicit actor in the model.

These steps have been defined in previous chapters where a process to reduce the abstraction level of a “pure” organizational model that uses goal-based elicitation processes and pattern language methods has been proposed. As a result of these previous analyses, an organizational model that includes the SSA is created. The organizational model that is extended with the SSA is the basis to generate the requirements for the system-to-be.

The use of an intermediate model (organizational model with the SSA) is one of the differences of the proposed method with current research works in the area, where the software requirements are directly generated form business functionalities.

It is important to point out that the generation of the requirements model is a simple process based on model transformation rules. This transformation is possible because the level of the intermediate model is closer to the requirements model. In this sense, the intermediate model represents the expected functionalities of the system-to-be. This is also the kind of information that is represented in a UML requirements model. Thus, we consider that both models represent the same information but represented in a completely different manner, one using UML and the other using the Tropos Framework.

Chapter 8

Case Studies

This Chapter details the case studies carried out in this research work in order to validate the proposed methodology to obtain requirements and conceptual models from organizational models.

8.1 Introduction

The rules, methods and patterns proposed in this thesis have been validated with three real case studies:

- Golf Tournaments Management
- Car Rental
- Technical Meeting Management

First two case studies were developed in the academic domain, whereas the last case study was developed in the context of industrial projects developed in the CARE Technologies enterprise. The application of the case studies was the source of most of the ideas presented in this thesis. The case studies were used to evaluate and to improve the proposed methods. This is why, in this Chapter we show the evolution of the proposed approach to generate requirements and conceptual models, pointing the lessons learned in terms of the advantages and issues of the analyzed methods.
The case studies developments are discussed in following sections. Syntheses of each project are followed by discussions of the methodology and the lessons learned through the application of the method.

8.2 Description of the case studies

8.2.1 Technical Meeting Management

The *Technical Meeting Management* was developed in the Valencia University of Technology in the Department of Information Systems and Computation, which organized the Workshop Requirements Engineering (WER´02) in year 2002. The aim of this case study was to model the organizational processes associated to the organization of a technical meeting and also the process to review papers to be presented in the meeting.

8.2.1.1 Description of the case study

The purpose of this case study was the analysis of the organizational context in order to obtain the requirements specification of a software system which handles the process of submission, assignament, evaluation and selection of papers for a conference.
In this case study, the organizational behavior needed to manage the technical meeting was also modeled. In this case, following events were considered: participants lodging management, participant’s transportation management, participant’s meal management, management of the proceedings generation, etc.

8.2.1.2 Methodology and case study artifacts

The *Technical Meeting Management* case study was conducted over a 5-month period. The average size of the models generated by students of a doctorate course is the following: 12 actors, 55 dependencies, 70 actors´ internal activities.

The first step carried out in this case study was the definition of the current situation of the enterprise using the *i** Framework. Figure 8.1 shows a fragment of the strategic rationale model[1] for this case study. This fragment defines the review process for the technical meeting. Some events in this case study are the following: the Chair of the Program Committee (*PcChair*) determines the topics of interest and selects the members of program committee (*PcMember*). The members can delegate the responsibility of reviewing to additional reviewers (*Reviewers*). Finally, the *PcMembers* and *Reviewers* send the evaluations to the *PcChair* indicating acceptance or rejection.

The next step of the process consists of determining the type of interaction of each organizational actor model with the SSA, which is called *Conference Review system*. An important concept used in this process is "*module*". A *module* represented the set of tasks performed by the actor to satisfy its goals with another actor. The *modules* were represented by internal task-refinement trees in the actors of the strategic rationale model. An actor could have more than one *module*; this indicated that the actor should fulfill more than one goal in the organizational model. In the case study, the *PcChair* had the modules: *assign paper to adequate PcMembers, obtain the highest number of quality papers, obtain quality reviews and send notifications* and *reviews to the Authors* (Figure 8.1).

[1] At the beginning of our research work , we started working with the *i** framework

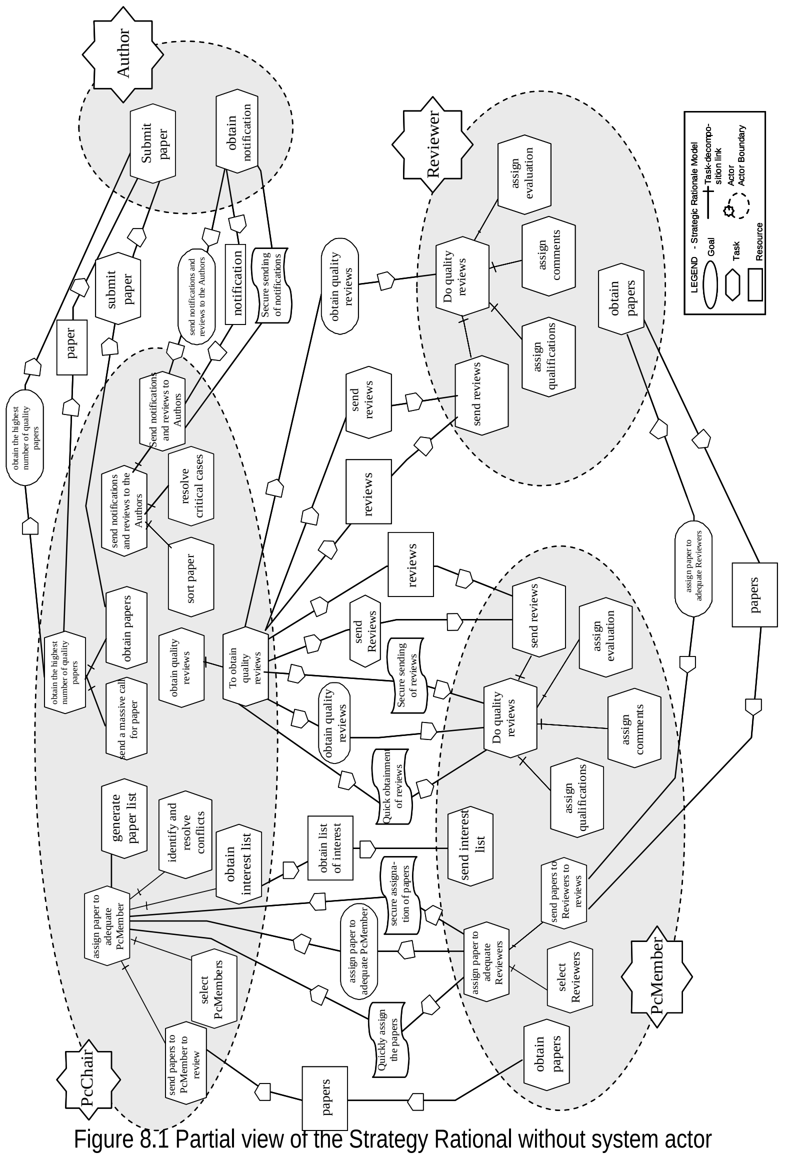

Author
Submit paper
obtain notification
submit paper
paper
send notifications and reviews to the Authors
notification
Secure sending of notifications
obtain the highest number of quality papers
Reviewer
assign evaluation
assign comments
Do quality reviews
assign qualifications
obtain papers
send reviews
obtain quality reviews
send reviews
reviews
LEGEND - Strategic Rationale Model
Goal
Task
Resource
Task-decomposition link
Actor
Actor Boundary
Send notifications and reviews to Authors
resolve critical cases
send notifications and reviews to the Authors
sort paper
obtain papers
obtain the highest number of quality papers
send a massive call for paper
obtain quality reviews
To obtain quality reviews
reviews
send Reviews
Secure sending of reviews
obtain quality reviews
Quick obtainment of reviews
send reviews
Do quality reviews
assign evaluation
assign comments
assign qualifications
assign paper to adequate Reviewers
papers
generate paper list
identify and resolve conflicts
obtain interest list
obtain list of interest
send interest list
secure assignation of papers
send papers to Reviewers to reviews
assign paper to adequate PcMember
assign paper to adequate PcMember
assign paper to adequate Reviewers
select Reviewers
select PcMembers
Quickly assign the papers
send papers to PcMember to review
papers
obtain papers
PcMember
PcChair

Figure 8.1 Partial view of the Strategy Rational without system actor

At this stage, we applied the proposed guidelines (presented in Chapter 4) to insert the system actor into the organizational model. As a result of the process of selection of the relevant information, a new organizational model is created (Figure 8.2) that represents: a) the actors with dependencies with the software system actor (SSA), b) the resources and task dependencies between the organizational actor and the SSA, and finally, c) the goal dependencies that have been derived in task and resource dependencies between the organizational actors and the SSA.

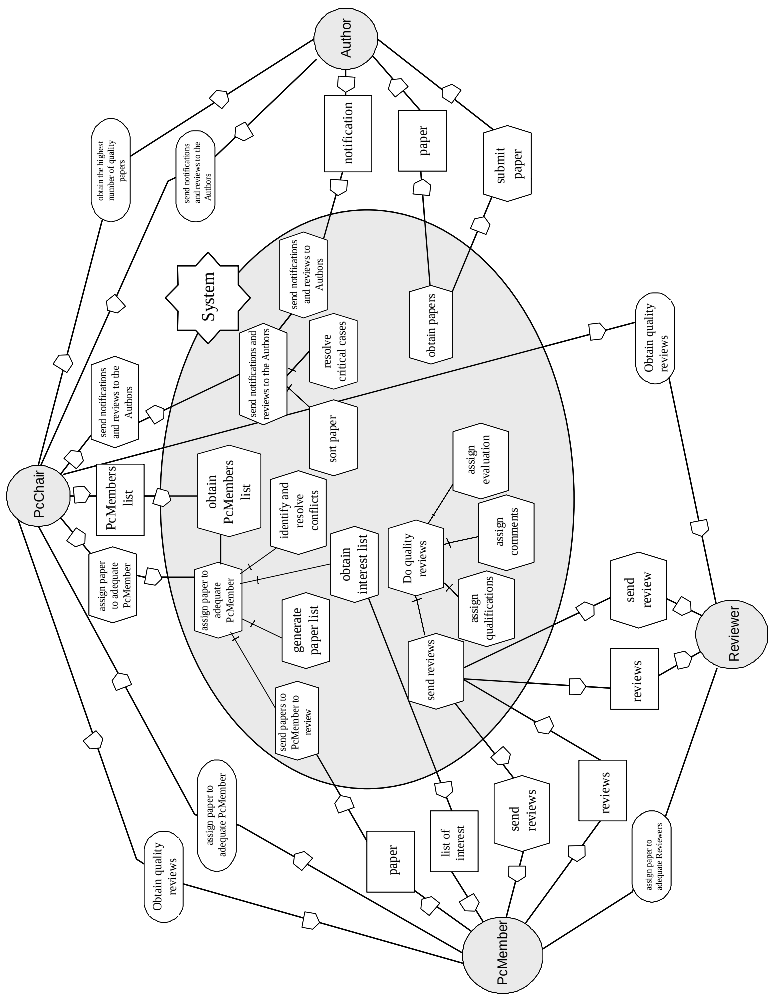

Author
PcChair
Reviewer
PcMember
System
notification
paper
submit paper
obtain the highest number of quality papers
send notifications and reviews to the Authors
send notifications and reviews to Authors
send notifications and reviews to the Authors
resolve critical cases
obtain papers
Obtain quality reviews
sort paper
send notifications and reviews to the Authors
PcMembers list
obtain PcMembers list
identify and resolve conflicts
assign evaluation
assign comments
Do quality reviews
obtain interest list
assign paper to adequate PcMember
assign paper to adequate PcMember
assign qualifications
send review
generate paper list
send reviews
reviews
send papers to PcMember to review
assign paper to adequate PcMember
Obtain quality reviews
paper
list of interest
send reviews
reviews
assign paper to adequate Reviewers

Figure 8.2 Insertion of the SSA into the organizational model

Once the relevant information has been selected and isolated in the SSA, our proposed method focuses on the translation process of the new organizational models in a use case model specification.

Therefore, the proposed steps were used to establish the correspondence between the elements of the organizational model specified in the *i** framework (with the explicit representation of the system actor) and the use case model and the corresponding scenarios specified in UML.
The result of the application of the transformational steps [Estr03a] [Past02] to the *Technical Meeting Management* case study is shown in Figure 8.3, which represents a fragment of the generated the use case model.

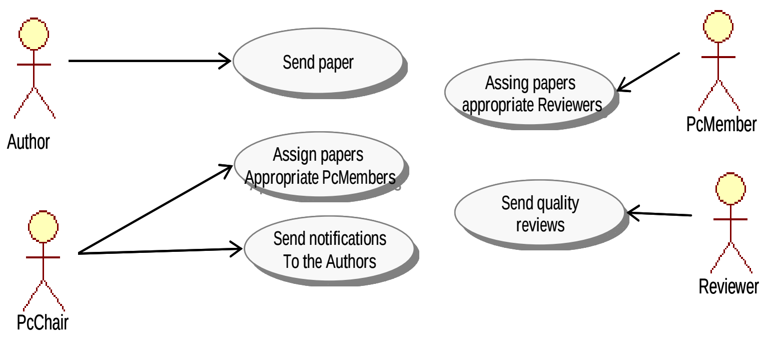

Author
Send paper
Assing papers appropriate Reviewers
PcMember
Assign papers Appropriate PcMembers
Send notifications To the Authors
Send quality reviews
Reviewer
PcChair

Figure 8.3 Partial view of the use case model

The representation of use case scenarios was done using a variant of the template proposed by L. Constantine [Cons99]. The Constantine template was used to show a sequence of events between the actors and the system. Table 8.1 illustrates an example for the use case *Send Reviews*.

Table 8.1 Specification of the use case *Send Review*

Use Case Name: Send Review		
Include:	None	
Extend:	None	
Preconditions	The PcMember has logged into the system.	
Post conditions	The system saves a new review information	
Primary Actor	PcMember	
Secondary Actors	None	
Roles	Reviewer	
User intentions		System responsibilities
1. The *PcMember* selects “Enter Review”		2. The system asks for Paper ID.
3. The *PcMember* introduces Paper ID		4. The system verifies paper ID and the system displays paper title.
		5. The system asks for review data
6. The *PcMember* introduces values for Originality, Technical Quality, Relevance and Overall Rating.		7. The system asks for reviewer’s comments
8. The *PcMember* introduces Author and Program Committee comments, and selects “Apply”.		9. The system saves the review information.
Asynchronous extensions		
The Reviewer can select *Stop* at any point		
Synchronous extensions		
If there is no paper ID, the system displays an error message at point 4.		

Another of the objectives of the proposed method is the direct generation of a conceptual model from the organizational specification.
In developing this case study, a preliminary version of the transformational rules was used to generate a conceptual schema from an organizational model that contains the SSA. This first version of the rules takes the organizational model to directly

generate the conceptual model. In the current version of the transformation method, the conceptual model is generated from the concerned object model obtained from the organizational model. At the end of this section, some conclusions are outlined to indicate why this first approach was not an appropriate means to achieve the thesis objectives.

Translation of Actors

The actors that participate in dependency relationships with the system actor are represented as classes in the conceptual schema.
The identification attribute of the actors of the organizational model is used to create the identification attribute of the classes, which represent these actors. The rest of the actors´ attributes are defined as variable attributes in the specification of the classes. This is a consequence of the lack of information of the organizational model to determine the stability of its attributes. For this reason, it is not possible to carry out a distinction between constant and variable attributes. In our case study, for example, the attributes *PcMemberId* and *AuthorId* are used to create the identification attribute of the classes *PcMember* and *Author*.
The mechanisms of creation and destruction of instances as well as the mechanisms of modification of variable attributes are placed by default in the conceptual model specification.

Translation of resource dependencies

The resources of the organizational model are translated into classes of the conceptual schema. Their attributes are used to create the attributes of their corresponding classes in the conceptual model.
In this first version of the translation process, a constant attribute was included in the definition of the resource to indicate the existence dependency with other resources in the model.
The existence of a constant attribute in the description of a resource allows us to create a relationship between the class of the specified resource and the class of the resource which is specified as constant attribute. To determine the relationship type, it is necessary to determine whether the classes placed as a constant attribute is "part *of*" the class which contains it. In this case, the relationship is an

aggregation. In case where no strong relationships between classes can be detected, an association relationship must be specified.
In our case study, for example, the *Notification* appears as a constant attribute of the resource *Paper*. In this case, the *Notification* is part of the *Paper*. Therefore, an aggregation between the classes *Notification* and *Paper* is created. It must be pointed out that there is no information that allows us to identify the type of aggregation or association obtained from the resource dependencies. Table 8.2 shows the OASIS specification for the classes *Paper* and *Notification*.

Table 8.2 Specification of the resource *Notification* and *Paper* in OASIS Language

Complex class Paper aggregation of
Notification(inclusive, dynamic,
univalued, disjoint, strict, notnull);
identification
    PaperId: (PaperId);
constant_attributes
    PaperId: Nat;
end_class
	Class Notification
	identification
	    NotificationId: (NotificationId);
	    PaperId: (PaperId);
	constant_attributes
	    NotificationId: Nat; PaperId: Nat;
	end_class

Translation of links between actors in resource dependencies

In this preliminary version of the transformation rules, the actors and resources involved in dependency relationships are used to generate the associations among classes in the conceptual schema. In this approach, an association must be defined between an actor and the resource involved in a dependency relationship.
Figure 8.4 shows a partial graphical representation of the conceptual schema obtained from the translation process for the case study.

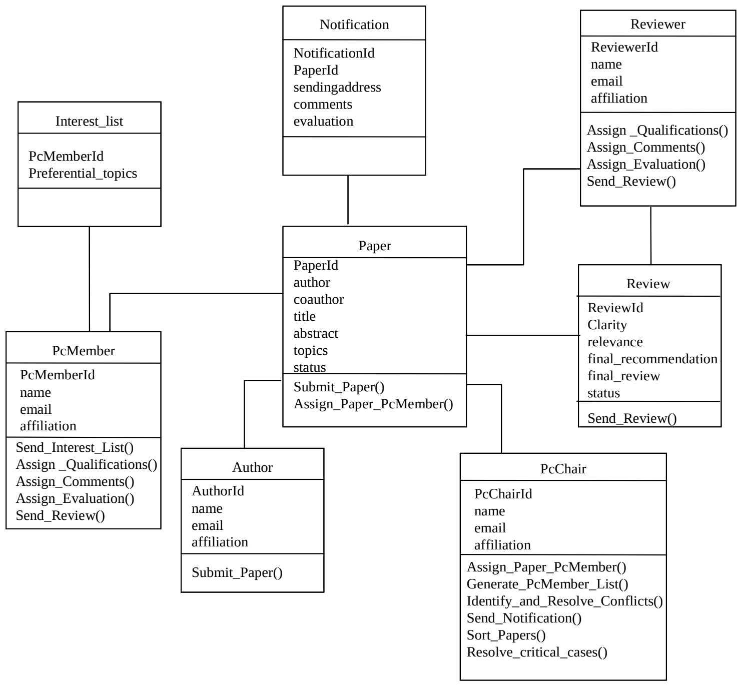


Figure 8.4 Conceptual Schema of the Conference Review System case study

8.2.1.3 Lessons learned

The *Technical Meeting Management* case study was the first real project developed in this research work. The application of the first version of the proposed method to generate requirements and conceptual schemas from an organizational model is the source of following lessons learned:

- One of the main conclusions of this work is the relevance of explicitly representing the SSA in the organizational model. This enables the analyst to isolate the behavior of the system in an individual actor. This behavior is the source of the candidate functions to be automated by the software-to-be. An initial set of guidelines were proposed to systematically carry out this process (see [Estr03b], [Estr02]).

- Other conclusion in this case study was the need to explore new ideas for generating the requirements model. In this first case study, which was published in [Past02] [Estr03a], a guided process to map the organizational model and the use case model was proposed. In this proposal, we defined a set of steps to establish the correspondence between the elements of the organizational model specified in the *i** framework (with the system actor integrated in explicit form) and the UML use case model and the corresponding scenarios. However, in this first version we have detected the need of an intermediate modeling stage to filter better the functionalities of the system-to-be.
- Other conclusions in this case study was the need to explore new ideas for generating the conceptual model. We have detected that with the application of the initial set of translation rules (published in [Mart03] [Mart04b]), we obtained the conceptual model of the organizational model, and not the conceptual model of the information system-to-be. This is because the relevant objects in this model belong to the business domain and these do not belong to the software system domain. In this context, if the generated conceptual model is implemented, the developed software system allows us to animate the behavior of the organizational model. We argue that an intermediate modeling stage is needed to obtain the relevant objects to define the system-to-be from the organizational model that contains the SSA. Therefore, the concerned objects will belong to the information system domain.

8.2.2 Golf Tournament Management

The *Golf Tournaments Management* (GTM) case study is a real project of the Care Technology Company.

8.2.2.1 Description of the case study

The objective of this case study was modeling an enterprise which is dedicated to manage and monitor Spanish golf tournaments. In the case study, the golf tournaments are validated by the Golf Federation, which ranks golfers in the golf championship. One of the main concerns of the golf organization enterprise is to provide partial results for each game. To do this, there are controllers that register the results of the golfers for specific holes. The results of the games must be validated by the Golf Federation to be considered as valid games.

The enterprise is responsible in partial and final classification of the games. The enterprise must also ensure the validity of the data of the games. To do this, representatives of the federation monitor all the games, and they play the role of “notary" of the results that the organization offers.

The professional’s golfer must register in different games. However, they could not be registered in all the games of a championship to obtain the final prize. In each golf field, the players compete in two games (typically in different journeys) and the best players compete in the final journey (unique). The best golfer in all journeys is the champion of the game.

The enterprise must register the results in each game in order to show partial and final results. The partial and final results generate news that need to be communicated on-line to the golfers.

8.2.2.2 Methodology and case study artifacts

This case study was developed by three development teams. The composition of the development teams was as follows: (i) Team 1 consisted of three expert analysts in the use of advanced tools for generating conceptual schemas from requirements models[1]; (ii) Team 2 included three expert analysts in the use of the CASE tool for automatically generating information systems from conceptual models[2]; (iii) Team 3 included two expert analysts in the use of *i** for organizational modeling.

This case study was conducted over a 9-month period. One of the objectives of this case study was evaluating the strengths and detected weaknesses of the framework used. Other of the objectives

[1] At the beginning of the evaluation, this team had limited knowledge of *i**.
[2] At the beginning of the evaluation, this team had no knowledge of *i**.

was to evaluate our methodological approach to generate requirements and conceptual models from organizational models.

The average size of the models generated by the development teams was: 8 organizational actors, 42 actors' dependencies, 103 actors´ internal activities.

In this case study, a second version of the transformation process was used in order to obtain a requirements and conceptual model for the system-to-be.

The evaluation of our proposal consists in building the organizational model for the case study, and to determine the tasks that required to be automated. In this context, the patterns of automation were identified and the translation rules were applied to the running example. Figure 8.5 shows a fragment of the organizational model for this case study. The shaded elements in Figure 8.5 were used to illustrate the proposed translation patterns. This model represents the actors who perform tasks in the business: the Organization (the company), the Golfers, and the Controllers and the Golf Federation. There are several dependencies among the actors: the Organization depends on Golfers to obtain the registration information for each player. The Golfers depend on the Organization to obtain a card with the game information. The Organization depends on the Controllers to get the partial results of each game. The Organization also depends on the Federation to validate the results of the games.

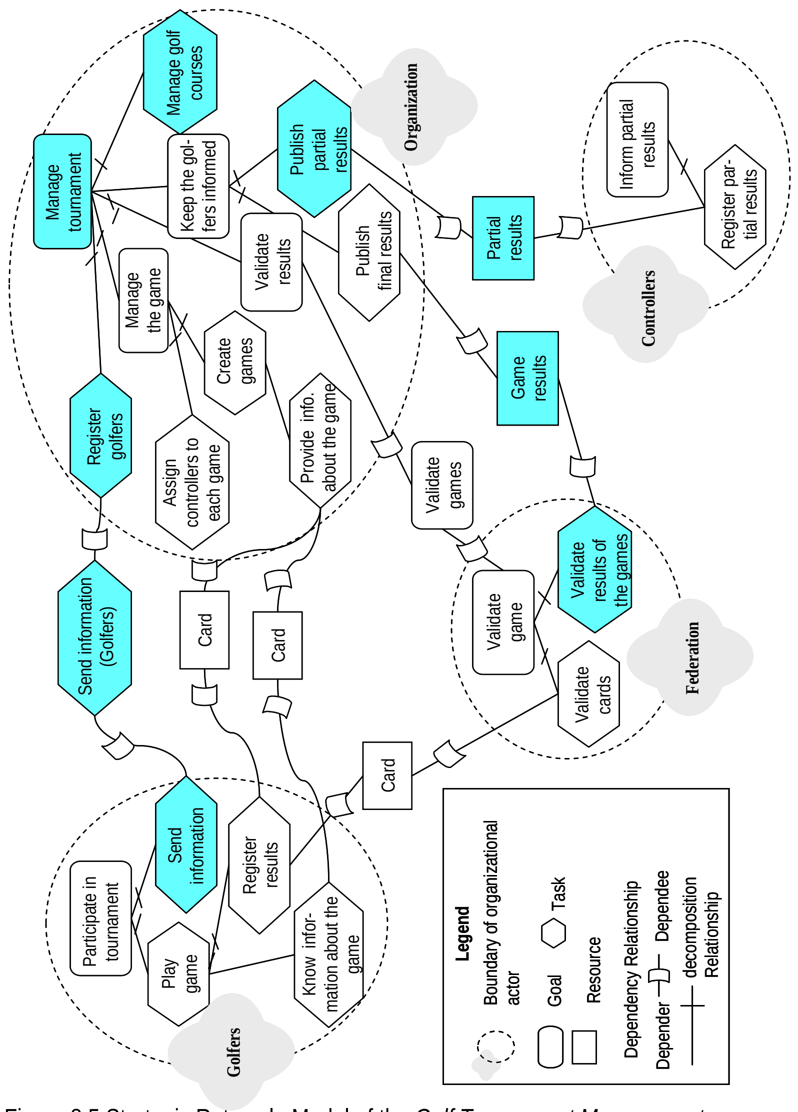


Figure 8.5 Strategic Rationale Model of the *Golf Tournament Management* case study

Once the strategic rationale model, which is shown in Figure 8.5, has been defined, we need to apply the proposed pattern language to make the model transformation systematic. Figure 8.5 shows an example of pattern application: The task *Register Golfers* of the *Organization* actor complies with the characteristics of the *Depender-Dependee Actor tasks* Automation Pattern. This is because this task was linked to the task dependency *send information (Golfers)* which also need to be automated. Once the pattern was applied, a task decomposition in the SSA was created (the parent node was the *Register Golfers* task, and the child node was the *obtaining information*). Both, a dependency relationship and an interaction relationship were also created. The results of the application of this pattern are shown in Figure 8.6. The dependencies that were modified or generated in this example are labeled with the number 1.

In other example, the task *Validate results of the games* of the *Organization* actor complies with the characteristics of the *Dependee Actor task* Automation Pattern. In this case, only the *dependee* actor task was automated. We applied the steps indicated for *the dependum is a resource*. The results of the application of this pattern are shown in Figure 8.6. The dependencies that were modified or generated in this example are labeled with the number 3. Another example is shown by the task *Publishing Partial Results* of this actor complied with the characteristics of the *Depender Actor Task* Automation Pattern. In this case, only the *depender* actor task was automated. We applied the steps indicated for *the dependum is a resource*. The results of the application of this pattern are shown in Figure 8.6. The dependencies that were modified or generated in this example are labeled with the number 2.

The *Final Task without dependencies* Automation Pattern is also discovered in the task *Manage Golf Courses* of organization actor. When the pattern was applied, this task was transferred to the SSA and a new dependency task between the *Organization* actor and the SSA was generated. This dependency allowed us to indicate that the Organization would provide the information about the golf courses to

the SSA. For this reason the element *Golf Courses* was placed as a parameter of the task dependency.
Finally, the goal *Golf Tournament Management* of the *Organization* actor complies with the characteristics of the *composite plan or composite goal* automation pattern. In this example, the tasks *Register Golfers*, *Publish partial results* and *Manage Golf Courses*; which are subtasks of the *Golf Tournament Management* Goal, have already been transferred to the SSA. For this reason, this goal was also transferred to the SSA. All this examples are shown in Figure 8.6. The paper [Mart04a] shows the rules of the pattern language proposed, where first version of the pattern language was used for carrying out this case study.
Another organizational model generated by using the pattern language is shown in Figure 8.7. This model was generated by the expert analysts in the use of advanced tools for generating conceptual schemas from requirements models (Team 1).

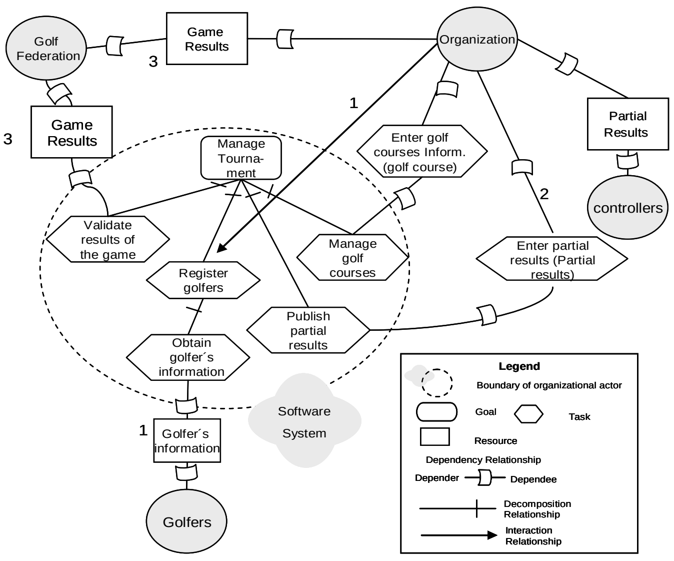


Figure 8.6 Organizational model generated by the application of the pattern language

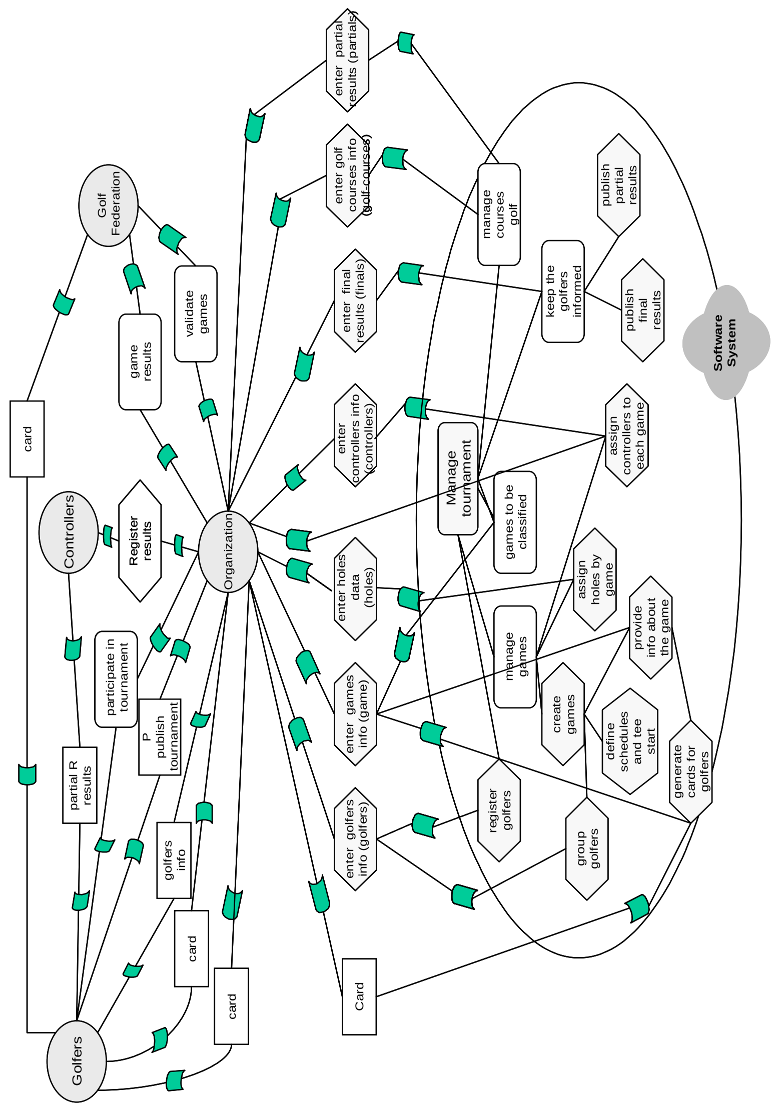


Figure 8.7 Another organizational model generated by the application of the pattern language

Once the organizational model with the SSA is obtained with the pattern language, the requirements model can be generated using this new organizational model. For carrying out this process the teams of development were focused on: a) the dependencies between the organizational actors and the SSA, and b) the internal tasks of the SSA.
Figure 8.8 shows all functional groups discovered using the transformational steps [Past02] [Estr03a] to the *Golf Tournament Management* case study.

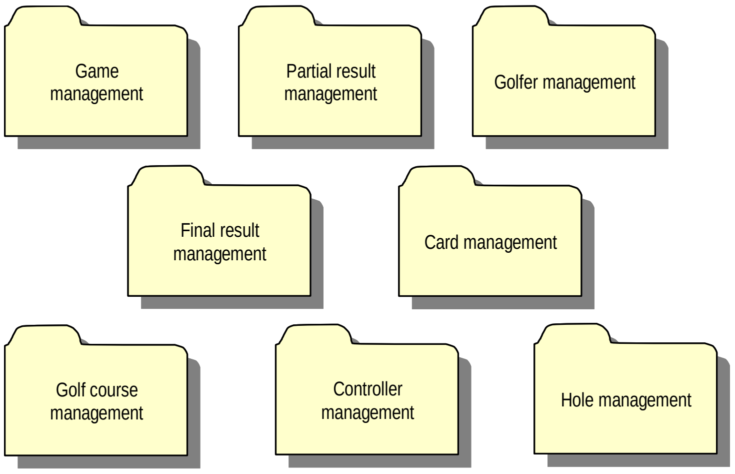


Figure 8.8 Example of functional groups discovered in the *Golf Tournament Management* case study

The use cases generated by default in each functional group are: creation, deletion and modification. They will permit us the manipulation of the resources through of the SSA.
The use case actors will be those that interact with the SSA. This interaction can be a dependency relationship between the organizational actors and the SSA.
Figure 8.9 shows the use cases detected for the functional groups in the case study.

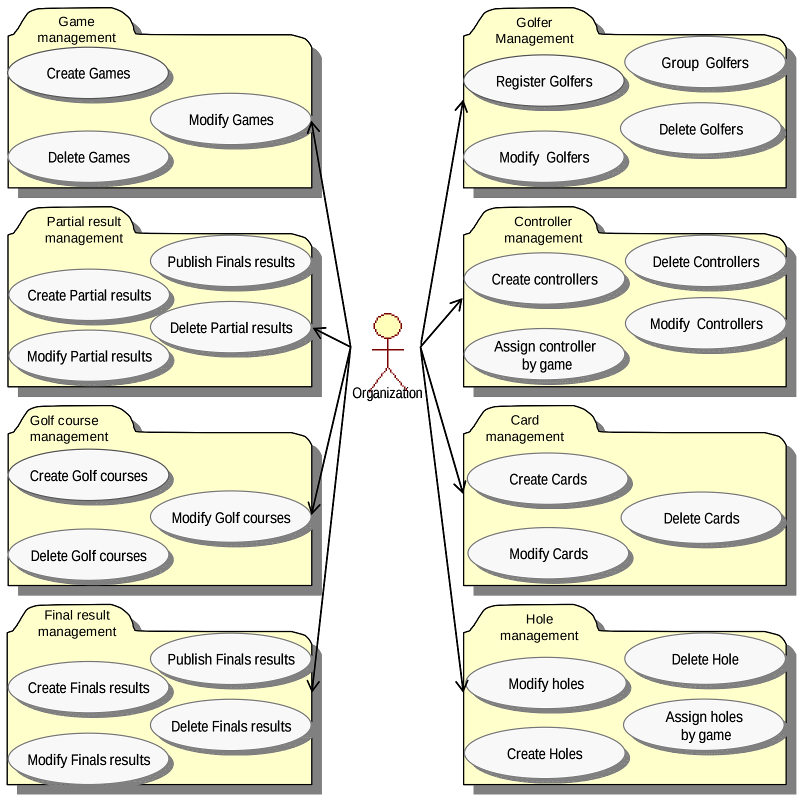


Figure 8.9 Functional groups and the use cases discovered in the *Golf Tournament Management* case study

Figure 8.10 shows the conceptual model obtained in the *Golf Tournament Management* case study. The rules used for generating this model were the shown in the [Past02] [Estr03a].

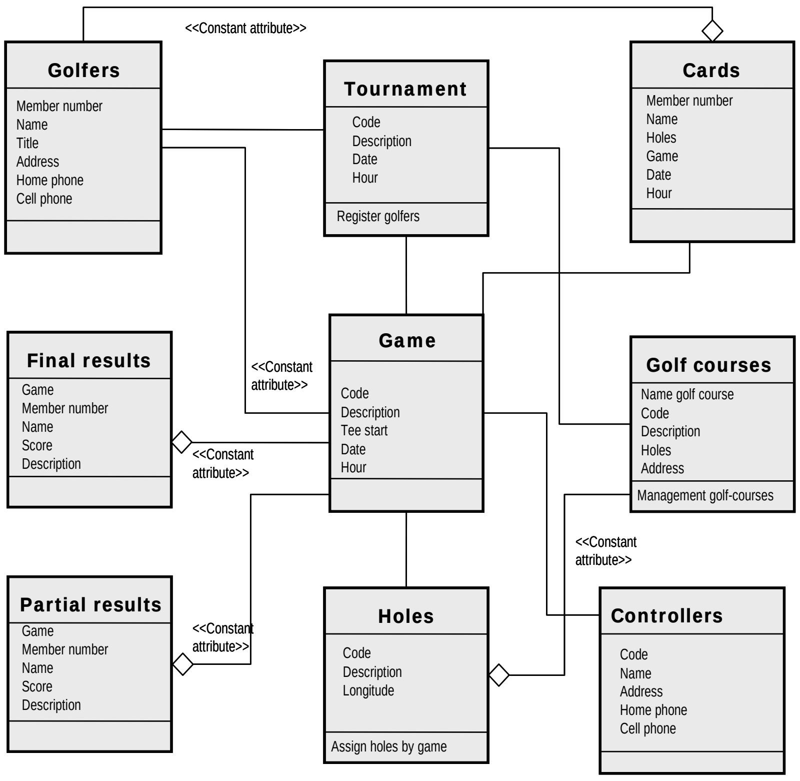


Figure 8.10 Partial view of the conceptual model of the *Golf Tournament Management* case study

8.2.2.3 Lessons learned

The *Golf Tournaments Management* (GTM) case study was the second real project developed in this PhD Thesis. The application of the second version of the proposed method to generate requirements and conceptual schemas from a organizational model was the source of following lessons learned:

- The proposed guidelines to insert the software system actor in the organizational model are not enough for a novel analyst. It is true that the guidelines offer a global idea of the process to generate the model, it is also true that they do not offer a systematic approach to carry out this process. This case study makes explicit the need to generate a set of patterns for inserting the SSA to the organizational model in a systematic manner.

- As a result of developing this case study, a set of patterns was developed besides the corresponding pattern language that indicates in the conditions in which the patterns need to be applied. The proposed pattern language considers all possibilities to delegate modeling elements from the organizational actors to the software system actor.

8.2.3 Car Rental

8.2.3.1 Description of case study

The objective of this case study was to model the processes of the *DENIA RentaCar* enterprise which is dedicated to manage car rental in Alicante, Spain. *RentaCar* has several branches in towns in several cities of Spain. These branches are located in tourist areas, and the set of cars to be rented have variations depending on two well differentiated seasons: winter and summer.
The cars are usually bought at the beginning of the season and sold, at the end of it. The purchase operations of the cars usually take into account their sale after a certain period of time (six months).
The main activity, the rental, involves other kinds of derived activities such as the car maintenance and repair, and extra rentals (telephone, driver, etc.).

Purchase & sale
At each branch cars, classified by car group, are available for rental. Only cars on the authorized list can be purchased.
Every new car should be covered by an insurance policy.
The car can be sold agreeing a price and a date of delivery. Cars are to be sold when they reach one year or 40,000 kilometers, whichever occurs first.

Employees
Each branch has a manager and users company. The Users can perform all habitual tasks of management (rent or return cars, change

rentals, maintain *Customer* data, maintain operations, etc.). The Manager can perform the same operations as users. However, they can also handle higher-level management tasks:

- Purchase, sale and eliminate cars
- Maintain rates (rental, insurances, and extras)
- Control insurance policies
- Manage employees

Rentals

Most rentals are through advance reservations; the rental period and the car group are specified at the time of reservation. *RentaCar* will also accept immediate ("walk-in") rentals, if cars are available.

At the end of each day, cars are assigned to reservations for the following day. If more cars have been requested than they are available in a group at a branch, the branch manager may ask other branch if they have cars they can transfer to him/her.

A car can be rented for different purpose, for example Tourism, Industrials, or to be reserved by some manager of the company.

A car from another branch may be allocated, if there is a suitable car available and there is time to transfer it to the pick-up branch. Some issues that must be taken into account are:

- Reservations may be accepted only up to the capacity of the pick-up branch on the pick-up day.
- If the *Customer* requesting the rental has been blacklisted, the rental must be refused 90 minutes after the scheduled pick-up
- A *Customer* may have multiple future reservations, but may have only one car at any time
- Only cars that are physically in *RentaCar* branches may be assigned.
- The end date of the rental must be before any schedule booking of the assigned car for maintenance or transfer.

Returns

Cars rented from one branch of *RentaCar* may be returned to a different branch. The renting branch must ensure that the car has been returned to some branch at the end of the rental period. If a car is returned to another branch than the one that rented it, ownership of the car is assigned to the new branch. If the car is returned late, an hourly charge is made up for a 6 hours´ delay; after 6 hours a whole day is charged. Some issues that must be taken into account are:

- If a car is returned to a location other than the agreed drop-off branch, a drop-off penalty is charged.
- The car must be checked for wear (brakes, light, tires exhaust, wiper, etc.) and damage, and repairs scheduled if necessary.
- If the car has been damaged during the rental and *Customer* is liable, the *Customer's* credit car company must be notified of a pending charge.

Servicing
For simplicity, only one booking per car daily is allowed. A rental or Extra service may cover several days. Also, the rent of a car can be with driver and phone. The rent of car must include insurance.

Customer
A *Customer* can have several reservations but only one car rented at a time. *RentaCar* keeps record of customers, their rentals and bad experiences such as late return, problems with the payment and damage to cars. This information is used to decide whether to approve a rental. Some issues that must be taken into account are:

- Each driver authorized to drive the car during a rental must have a valid driver's license.
- Each driver authorized to drive the car during a rental must be over 25.

Walk-in rentals
The end date of the rental must be before any schedule booking of the assigned car for maintenance or transfer.
If there are several available cars of the model or group requested, the one with the lowest mileage should be allocated. Some issues that must be taken into account are:

- If a rental request does not specify a particular car group or model, the default is group A (the lowest-cost group).

- Reservations may be accepted only up to the capacity of the pick-up branch on the pick-up day.
- If a specific model has been requested, a car of that model should be assigned if one is available. Otherwise, a car in the same group as the requested model should be assigned.
- If no specific model has been requested, any car in the requested group may be assigned.

Handover
The end date of the rental must be before any schedule booking of the assigned car for maintenance or transfer.
If there are several available cars of the model or group requested, the one with the lowest mileage should be allocated.
When a rental has been guaranteed by credit card and the car has not been picked up by the end of the schedule pick-up day, one day's rental is charged to the credit card and the car is released for use the following day. Some issues that must be taken into account are:

- The driver who signs the rental agreement must not currently have a car on rental.
- The car must not be handed over to a driver who appears to be under the influence of alcohol or drugs.
- The driver must be physically able to drive the car safely.
- The car must have been prepared –cleaned, full tank of fuel, oil and water topped up, tires properly inflated.
- The car must have checked for roadworthiness –tire treat depth, brake pedal and hand brake lever, travel lights, exhaust leaks, windscreen wipers.

Car maintenance & repairs

In this context, some issues that must be taken into account are:

- Each car must be Extra serviced every three months or 10,000 kilometers, whichever occurs first.
- A car that needs repairs must not be used for rentals.

8.2.3.2 Methodology and case study artifacts

This case study was conducted over a 9-month period. The average size of the models generated was: 13 actors, 143 dependencies, 219 actors´ internal activities.
This case study has been used as example along the document. Therefore, we have only included the final models obtained from applying the methodology proposed in this thesis to the case study.
Figure 8.11 shows a partial view of the actor diagram of the *Car Rental* case study where the SSA is included as a organizational actor. The actors involved in this diagram are: a) *Customer* who can play the role of person, b) Company Manager, c) User Company who can be the manager or Clerk actor, d) Mechanic, e) Insurance Company, f) Bank, g) Other branches and h) Car Rental System.

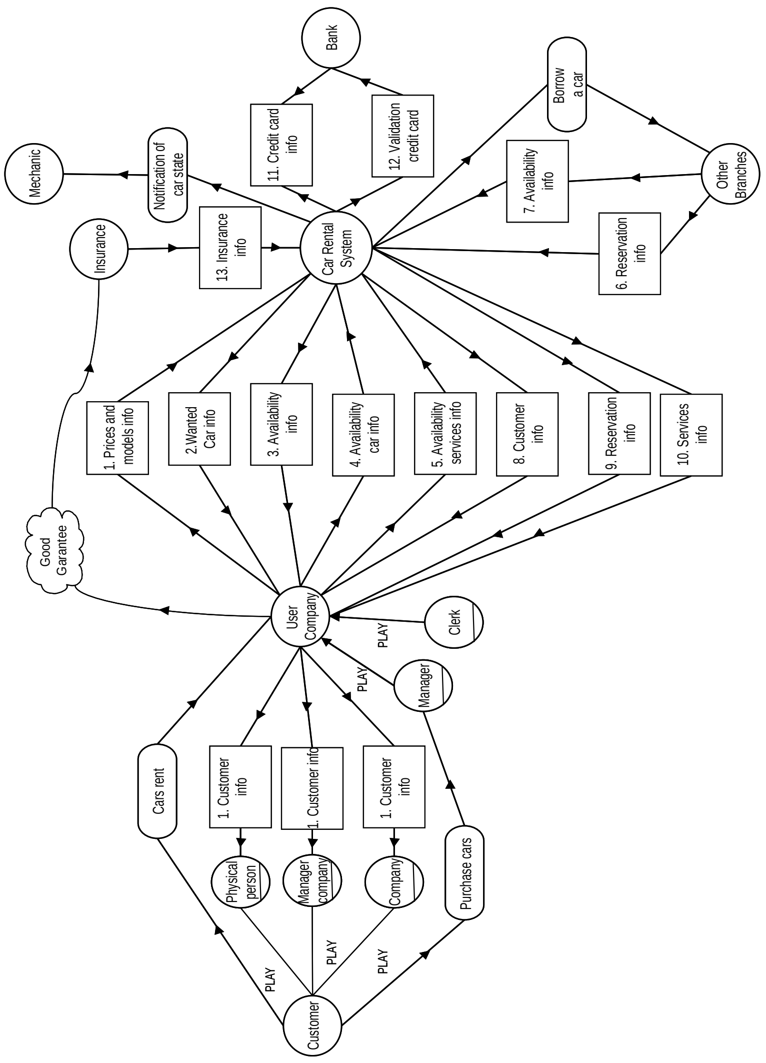


Figure 8.11 Actor diagram of the car rental case study

The goal diagram for the case study with the SSA is shown in Figure 8.12.

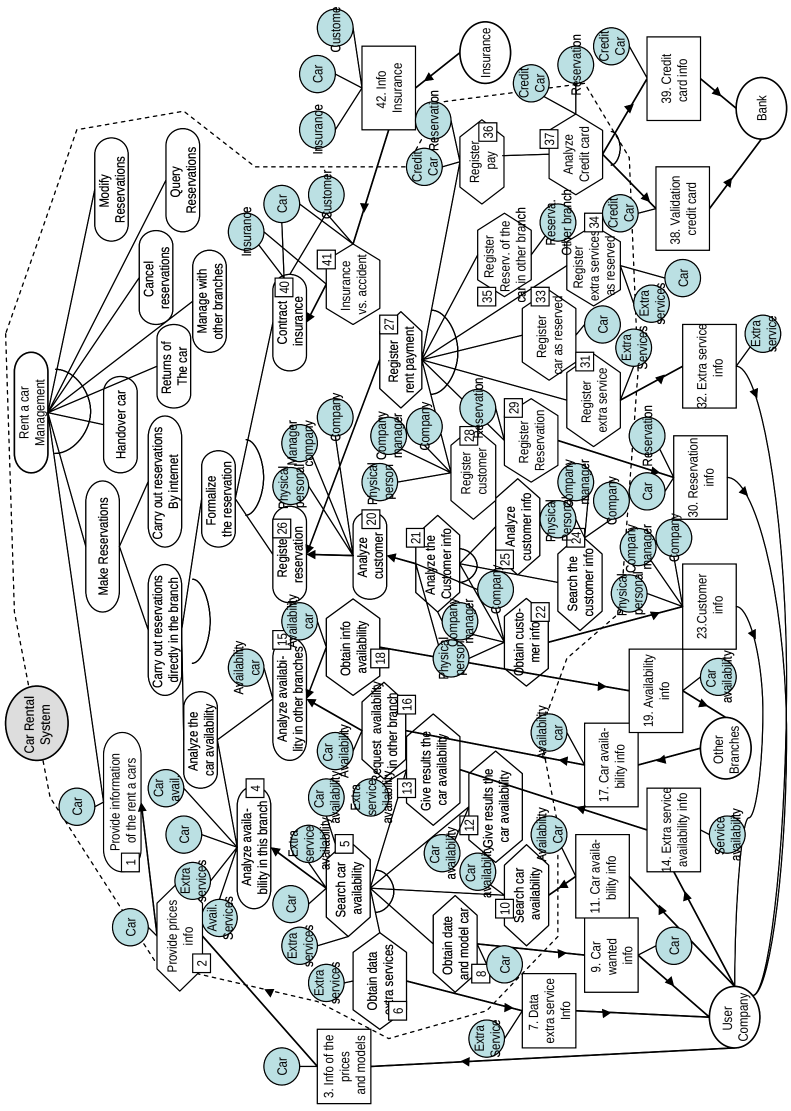


Figure 8.12 Partial view of the goal diagram for the *Car Rental* case study

The conceptual models generated from the organizational model are shown in Figure 8.13 and Figure 8.14, the first conceptual schema was generated taken into account the Optimization criterion. Therefore, the classes are generated in a global way, while Figure 8.14 shows the conceptual schema generated with the modularity criterion.

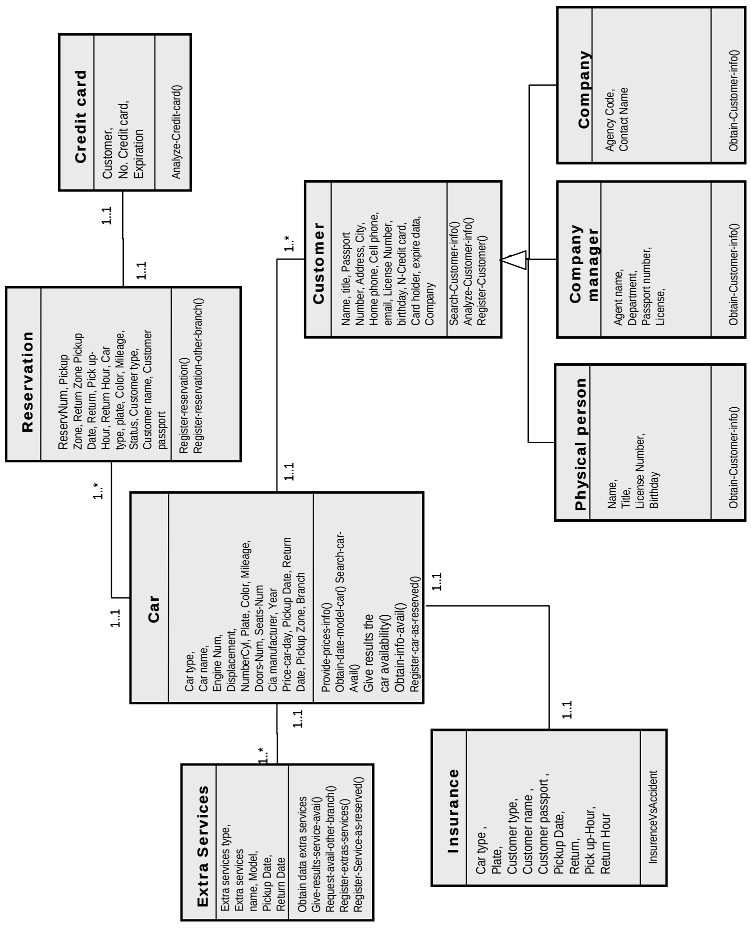

Credit card
Customer, No. Credit card, Expiration
Analyze-Credit-card()
Reservation
ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, plate, Color, Mileage, Status, Customer type, Customer name, Customer passport
Register-reservation()
Register-reservation-other-branch()
Customer
Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, birthday, N-Credit card, Card holder, expire data, Company
Search-Customer-info()
Analyze-Customer-info()
Register-Customer()
Company
Agency Code, Contact Name
Obtain-Customer-info()
Company manager
Agent name, Department, Passport number, License,
Obtain-Customer-info()
Physical person
Name, Title, License Number, Birthday
Obtain-Customer-info()
Car
Car type, Car name, Engine Num, Displacement, NumberCyl, Plate, Color, Mileage, Doors-Num, Seats-Num, Cia manufacturer, Year, Price-car-day, Pickup Date, Return Date, Pickup Zone, Branch
Provide-prices-info()
Obtain-date-model-car() Search-car-Avail()
Give results the car availability()
Obtain-info-avail()
Register-car-as-reserved()
Extra Services
Extra services type, Extra services name, Model, Pickup Date, Return Date
Obtain data extra services
Give-results-service-avail()
Request-avail-other-branch()
Register-extras-services()
Register-Service-as-reserved()
Insurance
Car type , Plate, Customer type, Customer name , Customer passport , Pickup Date, Return, Pick up-Hour, Return Hour
InsurenceVsAccident
1..1
1..1
1..*
1..*
1..1
1..1
1..1
1..1
1..1
1..*

Figure 8.13 Conceptual Model with the Optimization criteria

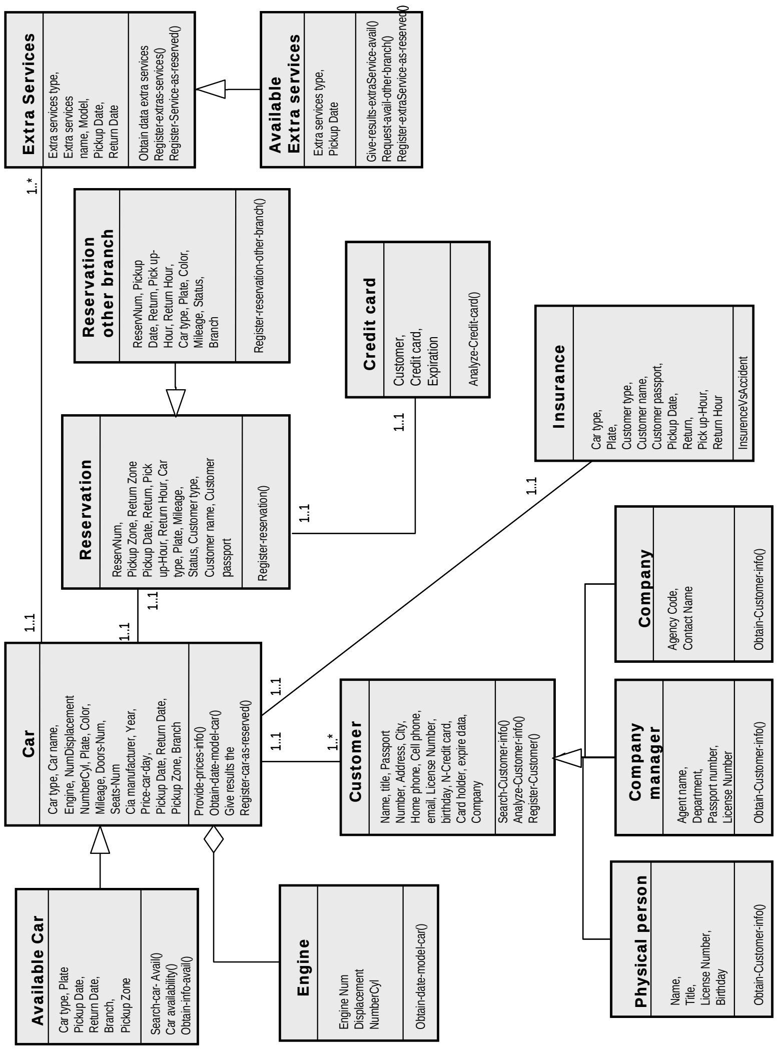

Extra Services
Extra services type, Extra services name, Model, Pickup Date, Return Date
Obtain data extra services
Register-extras-services()
Register-Service-as-reserved()
Available Extra services
Extra services type, Pickup Date
Give-results-extraService-avail()
Request-avail-other-branch()
Register-extraService-as-reserved()
Reservation other branch
ReservNum, Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Color, Mileage, Status, Branch
Register-reservation-other-branch()
Credit card
Customer, Credit card, Expiration
Analyze-Credit-card()
Insurance
Car type, Plate, Customer type, Customer name, Customer passport, Pickup Date, Return, Pick up-Hour, Return Hour
InsurenceVsAccident
Reservation
ReservNum, Pickup Zone, Return Zone Pickup Date, Return, Pick up-Hour, Return Hour, Car type, Plate, Mileage, Status, Customer type, Customer name, Customer passport
Register-reservation()
Company
Agency Code, Contact Name
Obtain-Customer-info()
Car
Car type, Car name, Engine, NumDisplacement NumberCyl, Plate, Color, Mileage, Doors-Num, Seats-Num Cia manufacturer, Year, Price-car-day, Pickup Date, Return Date, Pickup Zone, Branch
Provide-prices-info()
Obtain-date-model-car()
Give results the Register-car-as-reserved()
Customer
Name, title, Passport Number, Address, City, Home phone, Cell phone, email, License Number, birthday, N-Credit card, Card holder, expire data, Company
Search-Customer-info()
Analyze-Customer-info()
Register-Customer()
Company manager
Agent name, Department, Passport number, License Number
Obtain-Customer-info()
Available Car
Car type, Plate Pickup Date, Return Date, Branch, Pickup Zone
Search-car- Avail()
Car availability()
Obtain-info-avail()
Engine
Engine Num Displacement NumberCyl
Obtain-date-model-car()
Physical person
Name, Title, License Number, Birthday
Obtain-Customer-info()
1..*
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..*

Figure 8.14 Conceptual Model with the Modularity criteria

The requirements model generated from the organizational model is shown in Figure 8.15.

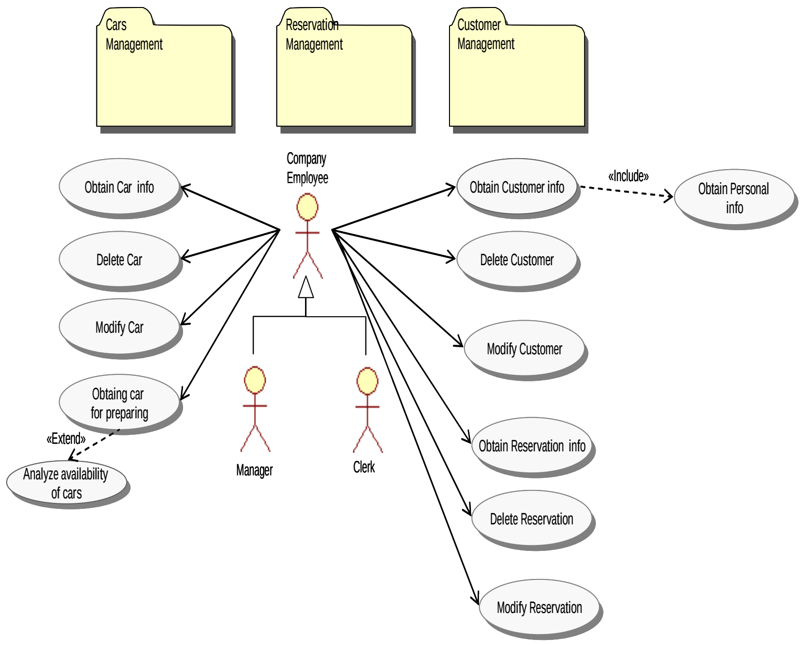


Figure 8.15 Partial view of the requirements model of the *Car Rental*

8.2.3.3 Lessons learned

This section summarizes the lessons that were learned from the *Car Rental* case study:

- This case study demonstrated the usefulness of an initial stage to detect the relevant tasks that need to be automated in order to better satisfy the organizational goals. Using this approach (which is based on organizational goal analysis) it was possible to better elicit the early requirements.
- This case study makes it explicit the usefulness of inserting the software system actor in the organizational model in order to isolate the expected functionality of the system-to-be.
- This case study demonstrates the need to consider an intermediate stage that can reduce the abstraction level of the late requirements specification to be closer to the design

stage. In our research work, a concerned object model is proposed in order to identify relevant elements for the information system domain.

- The intermediate model, which manages elements of the software domain, enables us to provide a systematic generation of an object-oriented conceptual model.
- The intermediate model also enables the analysts to generate a space of alternative conceptual models according to a specific optimization criterion.
- In developing this case study, we improve some rules and we add new rules to the proposed method to generate the requirements model. We also propose rules for building scenarios of the use cases according to RETO model. These method improvements have been detailed in Chapter 7.

8.3 Summary

The application of several case studies was very useful to demonstrate the applicability of the method. In fact, each case study enables us to improve the method until a stable state has been reached in the proposed method.

Chapter 9

Conclusions and Further Research

This chapter revises the stated research objectives and the main findings that can be drawn from this work. We also examine to what extent the research objectives have been met. The related publications derived from this research are also presented. Finally, we discuss the future research directions generated from this research work.

9.1 Conclusions

The contributions of this thesis are discussed by analyzing the research goals presented in Chapter 1, where three goals were proposed as a solution to the problems outlined:

- To reduce the abstraction level of a "*pure*" organizational model so that it is closer to the requirements model.
- To propose a methodological guide that allows a requirements model to be obtained from an organizational model.
- To propose a methodological guide that allows a conceptual model to be obtained from an organizational model.

These research goals were satisfied by developing three key processes of this thesis: the refinement process of the organizational model, the generation process of the conceptual models and the generation process of the requirements model.

9.1.1 The refinement process of the organizational model

The first goal of this thesis has been achieved by an initial elicitation process that allows the current situation of the enterprise to be represented and also permits the identification of the relevant tasks that need to be automated in order to better satisfy the organizational goals. A set of proposed steps that take into account quality factors, contradictions and contributions among plans has been proposed to identify the plans that must be automated through the system-to-be. This approach introduces a pattern language that allows us to consider all possible delegations of organizational plans to a new organizational actor that represents the software system-to-be.

In this process, the abstraction level of an organizational model is reduced by inserting the software system actor as an explicit actor in the organizational model. The model enables the analyst to focus specifically on describing the behavior of the system and on defining the dependency relationships of this actor with the rest of

organizational actors of the enterprise. Thus, it is possible to create a organizational model that is closer to the requirements model.
One of the key steps in this process is the generation of an intermediate model to identify the relevant information to generate the system-to-be in terms of concerned objects and relationships. We argue that the intermediate model represents the system requirements because it defines the expected functions of the system that are encapsulated in the boundary of the software system actor. One of the advantages of this approach is the explicit relationship between the organizational goals and the expected functionalities of the system. Thus, it is possible to justify the existence of a specific system function based on the achievement of objectives and goals.

9.1.2 The generation process of the conceptual models

The generation process of the conceptual models generates a space for alternative solutions in order to generate object-oriented conceptual models. This process is guided by a set of rules that define the steps needed in the definition of the conceptual model. The analyst manages the generation of the conceptual model according to specific optimization criteria. Therefore, instead of generating a unique conceptual model, this proposal enables the analyst to generate a model that is adapted to modularity, optimization, etc.
The second goal of this thesis has been satisfied by addressing the following sub-goals:

- Extending the organization model with monitoring plans and concerned objects. This is done in order to create a requirements specification that is closer to the system-to-be.
- Defining methodological guidelines that allow us to establish the correspondence between the organizational requirements that best satisfy the organizational goals and the conceptual model of the system.

In this thesis, we propose an extended organizational model with the concerned objects as an intermediate model between the late requirements specification and the conceptual model. The concerned object model permits the analyst to determine which elements are relevant in order to define the static structure of the system-to-be. This goal has been achieved by the generation process of the conceptual models, where the abstraction level of an organizational model is reduced with the identification of concerned objects, which allows us to define relevant elements in the information system domain.

9.1.3 The generation process of the requirements model

The generation process of the requirements model establishes a correspondence between the organizational requirements that best satisfy the organizational goals and the requirements of the system that is to be constructed.
The generation of the requirements model is guided by a set of rules that define the steps needed to transform the elements of the intermediate model (extended organizational model) into the elements of a UML use case model and its corresponding scenarios. It is important to point out that the source of the requirements generation process is the organizational model that explicitly contains the software system actor.
The third research goal of this thesis has been satisfied by addressing the following sub-goals:

- Developing guidelines that allow us to establish the correspondence between the late requirements model and a UML use case model.
- Developing guidelines to define the elements of the scenarios (primary actor, normal flow of action, preconditions, etc.) for each use case defined in the model

9.1.4 Using organizational model in the software production process

One of most interesting questions discussed in this thesis is the selection of the best alternative to use organizational models within the software production process. In the first stages of this work, it was difficult to give a clear answer to this question. This is because, in the initial research stages, we used a “pure” organizational model to make the model transformations, which is the same approach that current research works in this area uses. In this approach, the organizational model contains information from the organizational domain without references to the system to be developed. In this scenario, the generation of a late requirements model from the organizational model seems to be the best option because of the semantic distance between organizational concepts and conceptual modeling constructs.

However, in the current stage of our research work, it is possible to provide a better answer to the question of what the best way to use organizational model is: conceptual model generation is the best alternative, at least for the generation of the static part of the conceptual model. We support this with idea with the following: the organizational model and the concerned objects model that have been proposed in this thesis are equivalent to the requirements model for the system-to-be (late requirements). These intermediate transformation processes allows the analyst to represent: a) the expected functionality of the software system and b) the relevant objects (classes) to be considered in the definition of the static part of the conceptual model. Thus, the intermediate models reflect the same information that is presented in a requirements model: the expected functionality of the software system. This model represents the concerned objects as classes in the object model and the plans as the methods of the classes that have been generated.

For this reason, the transformation of the intermediate model into a late requirements specification (by using a use case model) is a simple mapping of concepts from both models.

We can conclude that, in our approach, the refinement steps are the adequate to directly generate the conceptual model instead of using late requirements models as the intermediate modeling stage.

9.1.5 Summary of contributions

Several contributions have been made in this thesis:

- **A method to identify the relevant plans to be automated**, which provides a set of rules to identify the relevant tasks[1] to be automated from the high-level goals of the stakeholders.
- **A pattern language,** which provides a systematic model transformation process between the organizational model and the model that explicitly includes the software system actor.
- **A method for inserting monitoring plans**, which provides a set of rules to detect undesirable behaviors in the system-to-be in order to take the corrective measures to manage them.
- **A method to define a space of alternative models**, which provides rules to define a conceptual model that fits a specific optimization criteria.
- **A method for linking late requirements with the *ONME* conceptual model generation,** which provides a set of rules and algorithms to obtain a conceptual model from an organizational model.
- **A method for linking late requirements with the *ONME* requirements model generation,** which provides a set of rules and algorithms to obtain a requirements model from an organizational model.

[1] The word "*relevant*" has been used in this thesis to indicate those elements whose automatic executions better satisfy the business goals.

9.2 Related Publications

The contributions of this thesis are supported by the set of publications carried out throughout this research work. These contributions have been published in three international journals, three book chapters, and thirteen conferences and workshops.

9.2.1 International Journals

- **Alicia Martínez**, Oscar Pastor, Hugo Estrada. “A pattern language to join early and late requirements”. *Journal of Computer Science and Technology (JCS&T), special issue on Software Requirements Engineering*. Vol. 5, No. 2. July 2005. ISSN 1666-6038.
- **Alicia Martínez,** Hugo Estrada, Oscar Pastor. “Generation of requirements model from organizational models: a pattern-based approach”, *Informatics Technology Management Journal* Num. 7, Vol. 2. December 2004. ISSN 1657-8236 pp. 11-21, (published in Spanish).
- Oscar Pastor, **Alicia Martinez Rebollar**, Hugo Estrada. “Generation of Software Requirements Specifications from Business Models”, *Informatics Technology Management Journal,* Num. 1, Vol. 1. 2002. ISSN 657-82364. pp. 53-65, (published in Spanish).

9.2.2 Book Chapters

- **A. Martinez**, O. Pastor, J. Mylopoulos, P. Giorgini. “Chapter 8 From Early to Late Requirements: A Goal-Based Approach”, Agent-Oriented Information Systems IV. Editor: Springer Berlin / Heidelberg. Volume: Volume 4898/2008. ISBN: 978-3-540-77989-6. February 2008.
- Oscar Pastor, Hugo Estrada, **Alicia Martínez**. *i**, its applications, variations, and extensions. The strengths and weaknesses of the *i** Framework: an experimental evaluation. Editors: (*Accepted for its publications by MIT Press*)

- J. Sanchez Diaz, O. Pastor Lopez, H. Estrada Esquivel, **A. Martinez Rebollar**, J. Belenguer Fáguas, "9. Semi Automatic Generation of User Interface Prototypes from Early Requirements Model"**,** Perspectives on Software Requirements Editors: Julio Cesar Sampaio do Prado Leite, Jorge Horacio Doorn. *Kluwer Academic Publishers, Boston Hardbound,* ISBN 1-4020-7625-8. USA 2004.

9.2.3 International Conferences and Workshops

- **Alicia Martínez**, Oscar Pastor, John Mylopoulos, Paolo Giorgini, "From Early Requirements to Late Requirements: A goal-based approach", in *Eight International Bi-Conference Workshop on Agent-Oriented Information System (AOIS-2006),* Luxembourg, Luxembourg, June, 2006.
- Hugo Estrada, **Alicia Martínez**, Oscar Pastor, John Mylopoulos, "An experimental evaluation of the *i** Framework in a Model-based Software Generation Environment", in *18th Conference on Advanced Information Systems Engineering (CAISE 06).* Luxembourg, Grand-Duchy of Luxembourg. June 2006. Lecture Notes in Computer Science, Vol. 4001, ISSN: 0302-9743. 2006. Pp. 513-527.
- **Alicia Martínez,** Oscar Pastor, Hugo Estrada. "A pattern language to join early and late requirements", in *VII Workshop on Requirements Engineering (WER 04),* Tandil Argentina 2004. pp 51-64.
- **Alicia Martinez**, Oscar Pastor, Hugo Estrada. "Isolating and specifying the relevant information of an organizational model: a process oriented toward information system generation", in *International Conference on Computational Science and its Applications (ICSSA 2004).* Perugia, Italy Springer LNCS 3046, pp. 783-790.
- Hugo Estrada, Oscar Pastor, **Alicia Martinez** and Jose Torres-Jimenez. "Using a Goal-Refinement Trees to obtain and refine organizational requirements", in *International*

Conference on Computational Science and its Applications (ICSSA 2004). Perugia, Italy, Springer LNCS 3046, pp. 506-513.

- Hugo Estrada, **Alicia Martinez**, Oscar Pastor. "Goal-based business modeling oriented towards late requirements generation", in *22nd International Conference on Conceptual Modeling (ER 2003)* October 2003, Chicago, Illinois, USA. ISBN 3-540-20-299-4, Springer LNCS 2813, pp. 277-290, 2003.
- **Alicia Martinez,** Jaelson Castro, Oscar Pastor, Hugo Estrada. "Closing the gap between Organizational Modeling and Information System Modeling", in *VI Workshop on Requirements Engineering (WER 2003)*. Piracicaba SP, Brazil, 2003. pp 93-108.
- Hugo Estrada, Jaelson Castro, Oscar Pastor, **Alicia Martínez.** "Goal-based organizational modeling oriented towards late requirements generation", in *17th Brazilian Symposium on Software Engineering - SBES'2003*.
- Hugo Estrada, **Alicia Martinez**, Oscar Pastor, Juan Sanchez. "Generation of Software Requirements Specifications from Business Models: a goal-based approach", in *V Workshop on Requirements Engineering (WER 2002).* Valencia, Spain November 11-12, 2002, pp. 177-193, (published in Spanish).
- **Alicia Martinez**, Hugo Estrada, Oscar Pastor. "The Business Model as starting point of the software requirements: a methodological approach", in *9° International Congress on Computer Science Research (CIICC´02).* Puebla, Mexico. October 2002, pp. 197-208, (published in Spanish).
- **Alicia Martinez**, Hugo Estrada, Juan Sanchez, Oscar Pastor. "From Early Requirements to User Interface Prototyping: A methodological approach", in *17th IEEE International Conference Automated Software Engineering (ASE2002).* Edinburgh, UK. September 2002, pp. 257-260.

- Hugo Estrada, **Alicia Martinez,** Oscar Pastor, Javier Ortiz, Erika Nieto**.** "Automatic generation of an Executable Conceptual Schema from a organizational model", in *V Iberoamerican Workshop Requirements Engineering and Software Environments (Ideas2002),* La Habana, Cuba, April 2002, pp. 281-292, (published in Spanish).
- Hugo Estrada E., **Alicia Martinez R.,** Oscar Pastor L., Javier Ortiz H., Octavio A. Rios T. "Automatic generation of a OO Conceptual Schema from a Work flow product model", in *IV Workshop on Requirements Engineering (WER2001).* National Technological University, Buenos Aires Argentina, November 2001, pp. 223-245, (published in Spanish).

9.3 Future research directions

With the modeling method proposed in this thesis, our intention is to give a further step in the process to integrate organizational modeling as an initial step the software production process. Our future works can be summarized as:

- Develop a prototype that automates the proposed method. At present several students are working in this direction.
- Develop a deeper analysis about of use of the monitoring plans.
  - Determine how the monitory plans can be used in the early requirements phase.
- Increase the strategies of the space alternatives to generate the conceptual models.

Bibliography

[Albe03] Albert M., Pelechano V., Fons J., Ruiz M., Pastor O. *Implementing UML Association, Aggregation, and Composition. A Particular Interpretation Based on a Multidimensional Framework.* In proceedings of the 15th Conference on Advanced Information Systems Engineering (CAISE 03). Klagenfurt, Austria, 2003. pp 143-158.

[Alen03] Alencar F. M. R., Pedroza F., Castro J., Ricardo C. O. Amorim. *New Mechanism for the Integration of Organizational Requirements and Object Oriented Modeling*. In proceedings of the VI Workshop on Requirements Engineering (WER 2003), Piracicaba, Brazil. November 2003. p.109-123.

[Alen00] Alencar, F., Castro, J., Cysneiros, G., Mylopoulos, J., *From Early Requirements Modeled by i* Technique to Later Requirements Modeled in Precise UML*, In Proceedings of the III Workshop on Requirements Engineering (WER 2000). Rio de Janeiro, 2000. pp 92-108.

[Alen99] Alencar F., *Mapping an Organizational Model in Precise specification*. Ph.D. Dissertation, Department of Informatics from University of Pernambuco. Recife, Brazil 1999.

[Alex02] Alexander, I. F., Stevens, S., *Writing Better requirements*, Addison-Wesley, 2002.

[Ambl03] Ambler Scott. Agile Database Techniques Effective Strategies for the Agile Software Developer. John Wiley & Sons. USA, November 2003.

[Alex79] Alexander C. *The Timeless way of Building*. Oxford University Press, New York, 1979.

[Alex77] Alexander C., Ishikawa S., and Silverstein M. A Pattern Language:towns, bulding, construction. Oxford University Press, 1977.

[Alpu05] Alpuente M., Gallardo M. M., Pimentel E., and Villanueva A. *A Semantic Framework for the Abstract Model Checking of tccp programs.* Theoretical Computer Science 346(1):58-95, 2005.

[Anto98] Anton, A. I. and Potts, C., *The use of goals to surface requirements for evolving systems*, in Proceeding of the International Conference on Software Engineering (ICSE 1998), 1998, pp. 157-166.

[Anto97] Antón I. Annie. *Goal Identification and Refinement in the Specification of Software-Based Information Systems*. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, June 1997.

[Anto96] Anton Annie. *Goal based requirements analysis*. In Proceeding of 2nd International Conference on Requirements Engineering ICRE'96, Colorado, USA, 1996. Pp. 136-144.

[Arau03] Araújo J., Coutinho P. *Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method.* 2nd International Conference on Aspect-Oriented Software Development, Boston, USA, March 2003.

[Beed97] Beedle Michael A. COOherentBPR –A pattern language to built agile organizations, Plop-97 Conference, 1997.

[Bide02] Bider Llia, and Khomyakov Maxim, *If you wish to change the world, Start with yourself. An Alternative Metaphor Objects Interaction.* In proceedings of the 4st International Conference on Enterprise Information Systems (ICEIS 2002), Ciudad Real, Spain 2002. Vol. 2, pp.732-742.

[Bock97] Bock B.Douglas. *Entity-Relationship Modeling and Normalization Errors.* In Journal of Database Management. IGI Publishing, Hershey, PA, USA, 1997, 8(1): 4-12.

[Bohe96] Boehm B. I*dentify quality-requirements conflicts.* In Proceedings of the 2nd International Conference on Requirements Engineering ICRE'96, Colorado spring, Colorado, 1996.

[Boeh78] Boehm B., Brown J.R., Kaspar H., Lipow M., McLeod G., and Merritt M. *Characteristics of Software Quality*. TRW Series of Software Technology. North Holland, Amsterdam, 1978.

[Booc99] Booch G., Rumbaugh J. and Jacobson I. *The unified Modeling Language Users Guide*. Addison-Wesly, 1999.

[Bres04] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., and Perini A., *TROPOS: An Agent-Oriented Software Development Methodology*. In Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers; May 2004, 8(3):203-236.

[Bube98] Bubenko J.A., Jr, D. Brash and J.Stirma. *EKD User Guide, Dept. of Computer and System Science*, KTH and Stockholm University, Sweden ESPRIT Project No. 22927, ELEKTRA, Document, Feb. 5, 1998.

[Bub95] Bubenko, J. A., Jr and Kirikova M., *Worlds in Requirements Acquisition and Modeling, in: Information Modeling and Knowledge Bases VI*. H. Kangassalo et al. (Eds.), IOS Press, Amsterdam, 1995, pp. 159-174.

[Bube94] Bubenko, J. A., Jr and M. Kirikova, *Worlds in Requirements Acquisition and Modeling*. In 4th European-Japanese Seminar on Information Modeling and Knowledge Bases, edited by K. Sweden, H. Kangassalo and B. Wangler, IOS Press, The Netherlands, 1994, pp. 159-174.

[Busc98] Buschmann, R. Meunier, H. Rohnert, P. Sommerland and M. Stal, Pattern - Oriented software Architecture: A system of Patterns. John Wiley & Sons, 1998.

[Cast02] Castro J. Kolp M. Mylopoulos J. *Towards Requirements-Driven Information Systems Engineering: The Tropos Project*. In Information System, Elsevier 2002, 27(2): 365-389.

[Cast01] Castro, J., Alencar, F. M. R., Cysneiro F., G., Mylopoulos, J. *Integrating Organizational Requirements and Object Oriented Modeling.* In proceeding of 5th IEEE International Symposium on Requirements Engineering. IEEE Press, 2001. pp.146-153.

[Cast98] Castelfranchi C. Modelling social action for AI agents. Artificial Intelligence. 103, 1998. pp. 157–182.

[Cesa02] Cesare S. Mark Lycett, *Business Modelling with UML, distilling directions for future research*. In proceedings of the Information Systems Analysis and Specification (ICEIS) Ciudad. Real, Spain, 2002, pp. 570-579.

[Chen76] Chen P.P. *The Entity Model –Toward a Unified View of Data.* ACM Transaction Database System 1976. 1(1): 9-36.

[Chun00] Chung, L., Nixon, B., Yu, E. and Mylopoulos,J. *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers 2000.

[Clar96] Clarke Edmund M. and Wing M. Jeannette. *Formal Methods: State of Art and Future Directions*. ACM Computing Surveys, December 1996, 28(4):626-643.

[Cock01] Cockburn Alistair, *Writing Effective Use Cases,* Addison-Wesley, USA, 2001.

[Cock97] Cockburn, Alistair. *Structuring Use Cases with Goals.* Journal of Object-Oriented Programming, Sep-Oct, 1997 and Nov-Dec, 1997. Also available on http://members.aol.com/acockburn/papers/usecases.htm

[Codd79] Codd E. F. *Extending the Database Relational Model to Capture More Meaning.* ACM Transaction Database System. 1979, 4(4): 397-434.

[Cohe97] Cohen D., Feather M.S., Narayanaswamy K., and Fickas S. *Automatic Monitoring of software Requirements.* In Proceedings of the 19th International Conference on Software Engineering (ICSE 1997), Boston, May 1997.

[Cons99] Constantine L.L; Lockwood L.A.D. Software for Use: A practical Guide to the Models and Methods of Usage-Centered Design. Addison Wesley 1999.

[Copl95] Coplien J. O., and Schmidt D. C., Eds. Pattern languages of program design, ACM Press/Addison-Wesley Publishing Co., New York, NY, 1995.

[Copl91] Coplien J. O.Advanced C++ Programming Styles and Idioms. Addison Wesley1991.

[Coss02] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. *Introducing pattern reuse in the design of multi-agent systems*. Agent, Infrastructure, Tools and Aplication (AITA'02) workshop at NODe02 Erfurt, Germany, October 2002. pp 8-9.

[Curt88] Curtis Bill., Kasner Herb, Iscoe Neil, *A Field Study of the Software Design Process for Large Systems*, Communications of the ACM, (31), 11, New York, USA, 1988, pp. 1268-1287.

[Dard03] Dardenne A., van Lamsweerde A., and Fickas S. *Goal directed requirements acquisition.* Science of Computer Programming, 2003. (20)1-2: 3-50.

[Dard93] Dardenne, A., van Lamsweerde A., and Fickas, S. *Goal Directed Requirements Acquisition*. Science of Computer Programming, vol. April 1993, Pp. 3-50.

[Dari96] Darimont, R. and van Lamsweerde, A. *Formal Refinement Patterns for Goal-Driven Requirements Elaboration*, in Proceeding of the 4th ACM Symposium on the Foundations of Software Engineering (FSE4), San Francisco, October 1996. pp. 179-190.

[Desa07] Desai. Anil. *Database Design.* Website http://mcpmag.com/columns/article.asp?EditorialsID=22 Last access December 2007.

[Dijk02] Dijkman Remco M., Joosten Stef M.M. *Deriving Use Case Diagrams from Business Process*. Technical Report TR-CTIT-02-08 Centre for Telematics and Information Technology, University of Twente, Enschede, 2002. ISSN 1381-3625.

[Domm99]Van Dommelen W., Joosten S., and de Mol M. *Comparative study to aids for managing business processes* (in dutch: Vergelijkend warenonderzoek hulpmiddelen beheersing bedrijfsprocessen. Technical report, Department of Finance, The Hague, 1999.

[Dorf90] Dorfman M., Thayer R.H. *System and Software Requirements Engineering*, IEEE Computer Society Press, 1990.

[Dull03] Dullea J., Chen R. Issues and Quality Measures for Database Design. Proceedings of 7th IASTED International Conference on Software Engineering And Applications (SEA2003) November 2003 Marina del Rey, CA, USA.

[Estr03a] Estrada H. Martínez A., Pastor O. *Goal based business modeling oriented towards late requirements generation*. In proceedings of the 22nd International Conference on

Conceptual Modeling (ER 2003). Vol. 2815, Chicago, Illinois USA, October 2003. Pp. 277-290.

[Estr03b] Estrada H. Castro J., Pastor O., and Martínez A. Goal-based organizational modeling oriented towards late requirements generation. In proceedings of the 17th Brazilian Symposium on Software Engineering (SBES'2003). Octubre 2003. Manaus, Amazonas, Brasil.

[Estr02] Estrada H. Martínez A., Pastor O., and Sánchez J. *Generation of Software Requirements Specifications from Business Models: a goal-based approach*. In Proceeding of the V Workshop on Requirements Engineering (WER 2002). Valencia, Spain November 2002, pp. 177-193

[Feat98] Feather M.S., Fickas S., Van Lamsweerde A., Ponsard C. *Reconciling System Requirements and Runtime Behaviour*. In Proceedings of the 9th International Workshop on Software Specification and Design IWSSD'98. IEEE. Isobe, Japan, April 1998.

[Fick95] Fickas S., and Feather M. *Requirements Monitoring in Dynamic Environments*. In Proceedings of the 2nd International Symposium on Requirements Engineering (RE'95), York, IEEE, 1995.

[Fick92] Fickas S. and Helm R., *Knowledge Representation and Reasoning. in the Design of Composite Systems*. IEEE Trans. On Software Engineering, June 1992. pp. 470-482.

[Fowl98] Fowler Martin. *Use and Abuse Cases.* Distributed computing. April 1998.

[Fowl97] Fowler M. *Analysis Patterns: Reusable Object Models*. Addison -Wesley 1997.

[Fran03] Frankel S. David, Model driven Architecture, applying MDA to enterprise computing, Wiley publishing, Inc. USA, 2003.

[Fuxm03] Fuxman Ariel, Liu Lin, Pistore Marco, Roveri Marco, Mylopoulos John: *Specifying and Analyzing Early Requirements: Some Experimental Results*. Proceeding of the 11th IEEE International Requirements Engineering Conference (RE'03), Monterey Bay, California USA 2003. Pages 105.114**.**

[Fuxm01] Fuxman Ariel, *Formal Analysis of Early Requirements Specifications*. MS Thesis. University of Toronto, Toronto, Canada, 2001.

[Gamm95] Gamma E., R. Helm, R. Johnson, J. Vlissides. Design Patterns, Addison-Wesley, 1995.

[Gamm94] Gamma E., Helm R., Johnson, R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Mass., 1994.

[Garz02] Garzas Javier and Mario G. Piattini Velthuis, *9 Patrones y refactorizaciones en la calidad software*. Editors. Mario G. Piattini, Felix O. García, *Calidad en el desarrollo y Mantenimiento de Software*. Ed. Ra-Ma. España 2002.

[Gior05] Giorgini P., Mylopoulos J., Sebastiani R.. *Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology*. In Engineering Applications of Artificial Intelligence, Elsevier 18(2), March 2005.

[Gior03] Giorgini P., Mouratidis H., Schumacher M. Security Patterns for Agent Systems. In proceeding of the Eighth European Conference on Pattern Languages of Programs. 2003.

[Gior02] Giorgini P., Mylopoulos J., Nicchiarelli E., and Sebastiani R. *Reasoning with Goal Models.* Proceedings of the 21st International Conference on conceptual Modeling (ER2002), Tampere, Finland, October 2002. LNCS-Springer Verlag.

[Gonz04] Gonzalez-Palacios Jorge and Luck Michael. *A Framework for patterns in Gaia: A case-study with Organisations.* In proceeding of the Agent-Oriented Software Engineering, 5th International workshop AOSE 2004. New York, NY, USA, July 2004. pp. 174-188.

[Gros01] Gross D., Yu E. *From Non-Functional Requirements to Design through Patterns.* Requirements Engineering. Springer-Verlag 2001, Vol. 6, pp. 18-36.

[Grun99] J. Grundy, *Aspect-oriented Requirements Engineering for Component-based Software Systems*, In proceedings of the 4th IEEE International Symposium on Requirements Engineering, IEEE Computer Society, Limerick, Ireland, 1999, pp. 84-91.

[Gues04] Guessoum Zahia, Ziane Mikal, Faci Nora. *Monitoring and Organizational-Level Adaptation of Multi-Agent System.* In Proceeding of the Autonomous Agents and Multi-Agent Systems (AAMAS2004). New York 2004. Pp.514-521.

[Hamm93]Hammer Michael and Champy James, *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Collins New York, USA, 1993.

[Haum98] Haumer, P., Pohl, K. & Weidenhaupt, K. R*equirements Elicitation andValidation with Real World Scenes*. IEEE Transactions on Software Engineering 24 (12), 1998. pp. 1036-1054.

[Hill99] Hilliard Rich. *Aspects, Concerns, Subjects, Views…**. In proceedings of the 14th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA´99). Workshop: Advanced Separation of Concerns, 1999.

[Horl01] Horling B., Benyo B., and Lesser V. *Using self-diagnosis to adapt organizational structures*. In 5th International Conference on Autonomous Agents. Montreal, 2001. ACM Press. Pp. 529-536.

[Hsia94] Hsia P., Samuel J., Gao J., Kung D., Toyoshima Y. and Chen C., *Formal Approach to Scenario Analysis.* IEEE Software, March-April, 1994. 11(2). pp. 33-41.

[Hull02] Hull, M.E.C, Jackson K., Dick, A.J.J., *Requirements Engineering*, Springer, 2002.

[IEEE03] The IEEE Computer Society. *Computing Curriculum, Software Engineering*. Public Draft 1. Software Engineering Education Knowledge Software, July 2003.

[Insf03] Insfran Emilio. A Requirements Engineering Approach for Object-Oriented Conceptual Modeling, PhD Thesis, Department of Information Systems and Computation, Valencia University of Technology, 2003.

[Ins02a] Insfrán E., Díaz I., Burbano M. *Modeling Requirements to Obtain Conceptual Models* (in Spanish). V Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software (IDEAS'2002). April 2002. pp 1-13.

[Insf02b] Insfrán E., Pastor O., Wieringa R. *Requirements Engineering-Based Conceptual Modelling.* In proceeding of the Requirements Engineering (RE), March 2002. **7**(2): 61-72.

[ISO01] International Standard ISO/IEC 9126: Quality Characteristics and Guide Lines for their use. Geneve (Switzerland), 2001.

[Jaco99] Jacobson I., Booch G., and Rumbaugh J. *The Unified Software Development Process*. Addison-Wesley, Reading, MA, 1999.

[Jaco95a] Jacobson Ivar, Ericsson Maria and Jacobson Agneta, *The Object Advantage: Business Process Reengineering with Object Technology.* ACM Press / Addison- Wesley Publishing Company, 1995.

[Jaco95b] Jacobson, I. & Christerson, P. *A Growing Consensus on Use Cases*. Journal of Object-Oriented Programming, 1995. 8(1): 15-19.

[Jaco94] Jacobson I., Ericsson M., and Jakobson A. *The Object Advantage: Business Process Reengineering with Object Oriented Technology*. Addison-Wesley, Reading, MA, 1994.

[Jaco92] Jacobson I., Christerson M., Jonsson P. and Overgaard G., *Object Oriented Software Engineering, a Use Case Driven Approach*. Addison Wesley. Reading, Massachusetts 1992.

[Jian06] Jiang L., Topaloglou T., Borgida A., Mylopoulos J. *Incorporating Goal Analysis in Database Design: A Case Study from Biological Data Management.* In proceedings of 14th IEEE International Requirements Engineering (RE 2006): Minneapolis, USA. Septermber 2006. Pp. 196-204.

[Kami02] Kaminka G. A., Pynadath D. V., and Tambe M. *Monitoring teams by overhearing: A multi-agent plan-recognition approach.* Journal of Intelligence Artificial Research, 17. 2002. pp. 83-135.

[Kiri94] Kirikova, M. and Bubenko, J. A. *Software Requirements Acquisition through Enterprise Modelling*. Stockholm University, Sweden. 1994.

[Kend99] E. Kendall. *Role models: Patterns of agent system analysis and design*. BT Technology Journal, October 1999, 17(4): 46–57.

[Klep03] Kleppe A., Warmer J., Bast W. MDA EXPLAINED, the model driven architecture: practice and promise, Addison Wesley, ISBN: 0-321-19442-X, Boston, April 2003.

[Kolp03] Kolp Manuel, Paolo Giorgini, John Mylopoulos. *Organizational Patterns for Early Requirements Analysis*. In proceeding of the 15th Conference on Advanced Information Systems Engineering (CAiSE '03), Klagenfurt/Velden, Austria 2003. Pp. 617-632.

[Kolp01] M. Kolp, J. Castro, and J. Mylopoulos. *A social organization perspective on software architectures.* In First Int. Workshop From Software Requirements to Architectures (STRAW'01), Toronto, Canada 2001. pp. 5-12.

[Kors99] Korson, T.: Constructing Useful Use Cases. http://software-architects.com/publications/korson/usecase3. (1999).

[Koto98] Kotonya G. and Somerville I. *Requirements Engineering process and techniques*, JhonWiley & Sons, USA 1998.

[Koub00] Koubarakis M. and Plexousakis D., *A Formal Model for Business Process Modeling and Design*. In proceeding of the 12th Conference on Advanced Information Systems Engineering CAiSE 2000, Stockholm, Sweden 2000, pp. 142-156.

[Kous04] Koushik Sen, Abhay Vardhan, Gul Agha, and and Grigore Rosu. *Efficient Decentralized Monitoring of Safety in Distributed Systems*. In proceeding of the 26th International Conference on Software Engineering (ICSE'04), IEEE, 2004.

[Kula03] Kulak Daryl Eamonn Guiney, *Use Cases requirements in context*. Addison-Wesley, USA, 2003.

[Lams01] Lamsweerde A. *Goal-Oriented Requirements Engineering: A Guided Tour.* Invited minitutorial, Proceeding 5th IEEE International Symposium on (RE'01), Toronto, IEEE, August 2001, pp. 249-263.

[Lams00] van Lamsweerde, A., *Requirements Engineering in the Year 00: A Research Perspective*. In Proceeding of the 22nd Int. Conference Software Engineering, Limerick, ACM Press, June 2000.

[Lams98] van Lamsweerde, A., Darimont, R., and Letier, E. *Managing Conflicts in Goal-Driven Requirements Engineering*. IEEE Transactions Software Engineering

24(11). Special Issue on Managing inconsistency in Software Development, November 1998. pp. 908-926.

[Lams95] van Lamsweerde, A., Darimont, R. and Massonet, P., *Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt*. In 2nd Int. Symposium on Requirements Engineering (RE'95), York, UK, March 1995. pp. 194-203.

[Larm01] Larman Craig. APPLYING UML AND PATTERN An introduction to Object-Oriented Analysis and Design and the Unified Process. 2nd Edition. Prentice Hall PTR. Upper Saddle River, NJ, USA 2001.

[Leit97] Leite, J.C.S.P. Rossi, G. Balaguer F. Maiorana V., *Enhancing a requirements baseline with scenario,* Proceedings of the Third IEEE International Symposium on Requirements Engineering RE97, IEEE Computer Society Press, (1997), pp. 44-53.

[Lete04] Letier E. and A. van Lamsweerde. *Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering.* Proceedings of FSE'04, 12th ACM International Symp. on the Foundations of Software Engineering, Newport Beach (CA), Nov. 2004, pp. 53-62.

[Louc98] Loucopoulos P., Kavakli V., Prekas N., Dimitromanolaki I., Yilmazturk C., Rolland C., Grosz G., Nurcan S., Beis D. and Vgontzas G., *The ELEKTRA project: Enterprise Knowledge Modelling for change in the distribution unit of Public Power Corporation.* In 2nd IMACS International, Conference on Circuits, Systems and Computers (IMACS-CSC'98). Athens, Greece 1998. pp. 352-357.

[Louc95] Loucupoulos, P. & E. Kavakli. *Enterprise Modelling and Teleological Approach to Requirements Engineering*, International Journal of Cooperative Information Systems, 4(1), March 1995. pp. 45-79.

[Luba93] M. Lubars, C. Potts and Richter. *A review of the state of the practice in requirements modelling.* In proceeding of IEEE International Symposium on requirements Engineering RE´93, pp. 2-14, San Diego California, January 1993. IEEE Computer Society Press.

[Magn00] Magnus Penker, Hans-Erik Eriksson. *Business Modeling With UML: Business Patterns at Work*. OMG Press John Wiley & Sons, ISBN: 0-471-29551. USA 2000.

[Maid04] Maiden, N., Jones, S., Manning, S., Greenwood, J. & Renou, L. (2004). *Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study*. In proceeding of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04) Riga, Latvia 2004. pp. 368-383.

[Mart04a] Martínez A., Pastor O., and Estrada H. *A pattern language to join early and late requirements.* In proceeding of the VII Workshop on Requirements Engineering (WER 04), Tandil Argentina 2004. ISBN: 950-658-147-9. pp 51-64.

[Mart04b] Martínez A., Pastor O., and Estrada H. *Isolating and specifying the relevant information of an organizational model: a process oriented toward information system generation.* In International Conference on Computational Science ans its Aplications (ICSSA 2004). Perugia, Italy Springer LNCS 3046, pp. 783-790.

[Mart03] Martinez A., Castro J., Pastor O., Estrada H. Closing the gap between Organizational Modeling and Information System Modeling, in *VI Workshop on Requirements Engineering (WER 2003)*. Piracicaba SP, Brasil, 2003. pp 93-108.

[Mart98] Martin Robert, Dirk Riehle, Frank Buschmann, and John Vlissides (eds). Patterns Languages of Program Design 3, Addison-Wesley, 1998.

[McDe94] McDermid John. *Software Engineer´s Reference Book.* Butterworth-Heinemann Ltd; New Ed edition. USA 1994.

[Mesz98] Meszaros G. and J. Doble, A Pattern Language for Pattern Writing, pages 529-574. Pattern Languages of Program Design 3, Addison Wesley, Robert Martin, D. Riehle and F. Buschmann ISBN 0-201-310112, USA, 1998.

[Moli03] Molina P. J., *User Interface Specification: From Requirements to Code Generation"*, PhD. Thesis, Department of Information Systems and Computation, Valencia University of Technology, 2003 (in Spanish).

[Mylo01] Mylopoulos, J., Kolp, M. and Castro, J., *UML for Agent-Oriented Software Development: The Tropos Proposal,* Proceeding of UML 2001.

[Mylo99] Mylopoulos, J., Chung, L., and Yu, E. *From Object-Oriented to Goal-Oriented Requirements Analysis*. Communications of the ACM, 42(1), January 1999. pp. 31-37.

[Nurc98] Nurcan S., Grosz G., and Souveyet C. *Describing business processes with a guided use case approach*. In Proceedings of the 1998 Conference on Advanced Information Systems Engineering, volume 1413 of Lecture Notes in Computer Science, pages 339–362, Berlin, 1998.

[Oliv07] OlivaNova Model Execution System. CARE Technologies. Retrieve November 3, 2005, from: http://www.care-t.com.

[Oliv03] Olivé Antoni, *Derivation Rules in Object-Oriented Conceptual Modeling Language*s. The 15th Conference on Advanced Information Systems Engineering (CAiSE '03), Klagenfurt/Velden, Austria 2003, pp. 404-420.

[OMG01] *OMG. Model Driven Architecture (MDA)* document number ormsc/2001-07-01, 2001.

[Orti01] Ortín M. J., García M. J., Moros B., Nicolás J. *El modelo de Negocios como base del Modelos de Requisitos: utilizando UML*. Jornadas de Ingeniería de Requisitos Aplicada, Sevilla, Junio, 2001.

[Past02] Pastor O., Martínez A., Estrada H. Generación de Especificaciones de Requisitos de Software a partir de Modelos de Negocios. *Gerencia Tecnológica e Informática,* Num. 1, Vol. 1. 2002. ISSN 657-82364. Pp. 53-65.

[Past01] Pastor O., Gómez J., Insfrán E. And Pelechado V., *The OO-Method approach for Information Systems Modeling: from Object-Oriented Conceptual Modeling to Automated* Programming. Information System, Elsevier, 200. 26(7): 507-534.

[Past99] Pastor O., Insfrán E., Pelechano V., *OO-Method: an software production environment combining conventional and formal methods*. In proceeding of the 9th Conference on Advanced Information System Engineering (CAISE 99), Barcelona, España, 1999.

[Past98] Pastor O., Pelechano V., Insfrán E., and Gómez J., *From Object-Oriented Conceptual Modeling to Automated Programming in Java.* 17th Int. Conference on Conceptual Modeling (ER´98), Singapore, 1998.

[Past97] Pastor O., Insfrán E., Pelechano V., Romero J. and Merseguer J. *OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods.* In proceding of the 9th Conference on Advanced Information Systems Engineering (CAiSE'97). Barcelona, Spain 1997.

[Past96] Pastor O., Pelechano V., Bonet B., Ramos I. *An OO, Methodological Approach for Making Automated Prototyping Feasible*. Proceedings of DEXA96, Springer-Verlag, 1996.

[Past95] Pastor O., Ramos I., *OASIS 2 (2.2): A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*, 3rd Edition, Servicio de Publicaciones, Technical University of Valencia, Spain, 1995.

[Pele01] Pelechado V., T*ratamiento de Relaciones Taxonómicas en entornos de producción Automática de software.* Una aproximación basada en patrones. PhD thesis, Department of Information Systems and Computation, Valencia University of Technology. December 2003.

[Pott94] Potts, C., K. Takahashi and Antón A. I. *Inquiry-Based Requirements Analysis.* IEEE Software, March 1994. pp. 21-32.

[Pres03] Pressman Roger S., *Software Engineering: A Practitioner's Approach*. McGraw-Hill, fifth edition, December 2003.

[Rati02] Rational Software, Business Modeling with the UML and Rational suite AnalystStudio, Rational software Corporation. From web site:

http://www.cs.unibo.it/people/faculty/cianca/wwwpages/labspo/BusinessModeling.pdf. September 2002.

[RE02] *RE'02*, IEEE International Requirements Engineering Conference, http://www.re02.org/

[Rieh96] Riehle D. Zllighoven H. *Understanding and Using Patterns in Software development.* Theory and practice of object Systems, 1996. 2(1):3-13.

[Robe99] Robertson S. And Robertson J., *Mastering the Requirements Process*, Addison Wesley, 1999.

[Robi04] Robinson N. William, Elfson Greg. *Goal Directed Analysis with Use Cases*. In Journal of Object Technology, 2004. 5(3): 125-142.

[Roll99a] Rolland R., Souveyet, C., Plihon, V., *Method Enhancement with Scenario Based Techniques*. In proceedings of the 11th International Conference on Advanced Information System Engineering (CAISE'99), Germany, 1999, pp 14-18**.**

[Roll99b] Rolland, C., and N. Prakash. From Conceptual modelling to requirements engineering. Technical report series 99-111, CREWS, 1999.

[Roll98c] Rolland, C., Souveyet, C. and Ben Achour, C., *Guiding goal modeling using scenarios*. IEEE Trans. Software Engineering, December 1998. 24(12): 1055–1071.

[Ross77] Ross D.T. and Schoman K.E. *Structured Analysis for Requirements Definition.* IEEE Transaction on Software Engineering, 1997. 3(1): 6-15.

[Rumb98a] Rumbaugh J., Blaha M., Premerlani W., F. Eddy, W. Lorensen, *Object Oriented Modeling and Design.* Prentice Hall. ISBN: 84-7829-037-0. Spain 1998**.**

[Rumb98b] Rumbaugh J., Jacobson I., and Booch G. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[Rumb91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Sanc03] Sanchez D. J. Validación de requisitos de usuario mediante técnicas de transformación de modelos y prototipación automática de interfaces de usuario. PhD Thesis, Department of Information Systems and Computation, Valencia University of Technology, February 2003.

[Saka80] Sakai Hirotaka. *Entity-Relationship approach to the Conceptual Schema Design*. ACM SIGMON Conference. Santa Monica, California 1980. pp 1-8.

[Samp05] Sampaio A., Rashid A., Rayson P. *Early-AIM: An approach for Identifying Aspects in Requitements.* The 13th IEEE International Requirements Engineering Conference (RE'05), September 2005, Paris France, pp.487-488.

[Sann02] Sannicolò F., Perini A., Giunchiglia F. *The Tropos Modeling Language. A user guide.* Technical Report #DIT-02-0061, University of Trento, 2002.

[Sant02] Santander V. F. A., Castro J.: *Deriving Use Cases from Organizational Modeling.* 10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE 2002), Essen, Germany. September 2002. pp. 32-42.

[Sant01] Santander V., Jaelson B. C., *Developing uses cases from organizational modelling.* Proceedings of the IV Workshop on Requirements Engineering (WER 2001), Buenos Aires Argentina, November 2001.

[Schm95] Schmidt D., Using design patterns to develop reusable object-oriented communication software. Special issue on object-oriented experiences and future trends, 38(10), ACM Press New York, NY USA, October 1995, pp. 65-74.

[Seli03] Selic B., *The pragmatics of Model-Driven Development,* IEEE Software, Vol. 20, No. 5, pp. 19-25, 2003.

[Serd91] Serdanas A. and Serdanas C. and Gouveia. *OBLOG Object-Oriented Logic: An informal introduction.* Technical Report, INESC, Lisbon (1991).

[Shar02] Sharma R. *Microsoft SQL Server 2000: A Guide to Enhancements and New Features.* Pearson Education; Pap/Cdr edition March 2002.

[Sich02] Sichman J. S. and Conte R. *Multi-agent dependence by dependence graphs.* In Proceeding of the Autonomous Agents and Multi-Agent Systems (AAMAS2002). Boulogna, Italy, 2002. ACM. pp 483–490.

[Some05] Stéphane S. Somé: Use Cases based Requirements Validation with Scenarios. The 13th IEEE International Requirements Engineering Conference (RE'05), September 2005, Paris France, pp. 465-466.

[Stee96] Steeg Martin. *The Conceptual Database Design Optimizer CoDO- Concepts, Implementation, Application.* In proceedings of the 15th International Conference on Conceptual Modeling (ER 1996). Vol. 1157, Cottbus, Germany, USA, October 1996. Pp. 105-120.

[Sutt04] Sutton M. Stanley Jr. and Rouvellou Isabelle. *Chapter 21: Concern Modeling for Aspect-Oriented Software Development.* Editors Robert E. Filman, Tzilla Elrad, Siobhán Clarke and Mehmet Aksit. *Aspect-Oriented Software Development.* Addison Wesley Profesional, October 06, 2004.

[Thay02] Thayer Richard H., Dorfman Merlin, Dixie Garr, *Software Engineering*. Volume 1, the Development Proces. IEEE Computer Society Press. Wiley-IEEE Computer Society Press Wiley Publisher. March 2002.

[UML07] OMG´s Issue Reporting Procedure. Unified Modeling Language: Superstructure, version 2.1.1. February 2007. http://www.omg.org/docs/formal/07-02-03.pdf

[UML99] *UML Specification*. http://www.rational.com/uml/index.jtmpl. We have referenced V1.3 Alpha R5, March 1999 in this paper. pp. 2.113- 2.123.

[Url06] Appleton Brad. Patterns and Software: Essential Concepts and Terminology.

http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html. Last access May 2006.

[WMod07] WIKIPEDIA The Free encyclopedia. Website: http://en.wikipedia.org/wiki/Modularity_%28disambiguation%29 Last access August 2007.

[WOpt07] WIKIPEDIA The Free encyclopedia. Website: http://en.wikipedia.org/wiki/Optimization. Last access August 2007.

[Yu00] Yu, E., Liu, L. *Modelling Trust in the i* Strategic Actors Framework*. Proceeding of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies at Agents2000, Barcelona, Spain, 2000, pp. 175-194.

[Yu98] Yu, 1998. *Why Agent-Oriented Requirements Engineering.* In proceeding of the 4th International Workshop on requirements Engineering: Foundations of Software Quality, Pisa, Italy E. Dubois, A.L. Opdahl, K. Polh, eds Presses Universitaires de Namur (1998) 15-22.

[Yu97] Yu Eric, *Towards Modelling and Reasoning support for Early-Phase Requirements Engineering*. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97) pp. 226-235 .IEEE Computer Society Washington, DC, USA 97.

[Yu95] Yu Eric, *Modelling Strategic Relationships for Process Reengineering*, PhD Thesis, University of Toronto, Toronto, Canada, 1995.

[Yu94] Yu S. K. Eric, Mylopoulos John. Toward Modelling Strategic ActorRelationship forInformation System Development –with Examples from Business Process Reengineering. 4th Workshop on Information Technologies and Systems (WITS´94), Vancuver Canada, December 1994.

[Zave97] Zave P. Classification of Research Efforts in Requirements Engineering. ACM Comput. Surv. (1997). 29(4): 315-321.

List of figures

List of tables

Glossary

Actor	An entity that has strategic goals and intentionality within the system or the organizational setting. An actor represents a physical or a software agent as well as a role or position.
***Atomic* Elements**	Those elements that do not need to be decomposed into other sub-elements.
Belief	The actor's knowledge of the world.
Capabilities	The ability of an actor of defining, choosing and executing a plan for the fulfillment of a goal, given certain world conditions and in presence of a specific event.
***Composite* Elements**	Those elements whose execution is carried out by decomposing it into other sub-elements.
Concerns	A concern expresses a specific interest in some topic pertaining to a particular system of interest (or other subject matter).
Contribution	It represents a relationship between goals or plans representing how and how much goals or plans can contribute, positively or negatively, in the fulfillment of the goal.

Decomposition	It represents a relationship between goals or plans representing AND/OR decomposition of root goal/plan into sub-goals/*subplans*.
Dependency	It represents a relation between two actors, which indicates that one actor depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource. The former actor is called the *depender*, while the latter is called the *dependee*. The object around which the dependency is centered is called *dependum*. Dependum can be either goal, or resource, or task.
External Elements	Those elements that are represented graphically in a dependency relationship as *dependum*.
Goal	It represents actors' strategic interests. There are distinguished *hard goals* from *soft goals*, the second having no clear-cut definition and/or criteria for deciding whether they are satisfied or not.
Internal Elements	Are those elements that are defined inside the boundary of an organizational actor.
Information system	A software system used to support the activities in a business.
Model Transformation	The process of converting one model to another model or the same system.

Monitoring	An on-going process of reviewing programs´s activities to determine whether set standards or requirements are being met.
Monitoring System	An on-going system to collect data on a programs activities and outputs, designed to provide feedback on whether the program is fulfilling its functions, addressing the targeted population, and/or producing those Extra services intended.
Pattern	The description of a general solution to a common problem or issue from which a detailed solution to a specific problem may be determined. Software development patterns come in many flavors, including but not limited to analysis patterns, design patterns and process patterns.
Plan	It represents, at an abstract level, a way of doing something. The execution of plan can be a means for satisfying a goal or a soft goal.
Requirement	A requirement species how a goal should be accomplished by a proposed system Requirements. It also represents the capabilities that a system must provide in order to satisfy the goals of stakeholders. An example of an operation requirement for the goal Corporate profits maximized is Customer calls should be handled in less than minutes with the theory that you can improve the cost-benet to the corporation.

Functional requirement	It describes the behavioral aspects of a system
Nonfunctional requirement	It describes the non behavioral aspects of a system capturing the properties and constraints under which a system must operate.
Resource	A physical or an informational entity.
Stakeholder	A Stakeholder is anyone who claims an interest in a given enterprise or systems Stakeholders are those individuals who can share information regarding the proposed system its implementation or the problem domain.
Use Case (in a information system)	A behaviorally related sequence of transactions performed by an actor in a dialogue with the system to provide some measurable value to the actor.
Use-Case Model	A set of use case, actor and their relations.

List of abbreviations and acronyms

CARE	Computer Aided Requirements Engineering
CASE	Computer Aided Software Engineering
FELRE	From Early-Late Requirements
KAOS	Knowledge Acquisition in an automated Specification
GBRAM	Goal-Based Requirements Analysis Method
GDRE	Goal-Directed Requirements Engineering
MAL	Modal Actions Logic
MS	Mission Statement
NFR	Non-Functional Requirement
OASIS	Open and Active Specifications of Information Systems
OOMETHOD	Object Oriented Methodology
OMT	Object Modeling Technique
ONME	OLIVA NOVA Model Execution
OO	Object Oriented
RE	Requirements Engineering
RETO	Requirements Engineering TOol
SE	Software Engineering
SSA	Software System Actor
UML	Unified Modeling Language
UPV	Universidad Politécnica de Valencia
YSM	Yourdon System Model